Teoría de la Complejidad Computacional

Tema 5: Problemas NP completos

David Orellana Martín

Grupo de Investigación en Computación Natural
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

dorellana@us.es

Máster Universitario en Matemáticas

Curso 2023-2024

Índice

Reducibilidad en tiempo polinomial

Sean X e Y problemas de decisión. Se trata de definir el siguiente concepto: "X es **más fácil** que Y" (resp., Y es **más difícil** que X).

- Formalizar la **dificultad comparativa** de la resolubilidad mecánica de un problema respecto de la de otro.

Definición: Sean $L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^*$ lenguajes. Diremos que L_1 **es reducible a** L_2 **en tiempo polinomial sii** existe una función total $g : \Sigma_1^* \to \Sigma_2^*$ tal que:

- g es computable en tiempo polinomial.
- $\forall u \in \Sigma_1^* \ (u \in L_1 \iff g(u) \in L_2)$.

En tal situación, notaremos $L_1 \leqslant^p L_2$ y diremos que g es una *reducción polinomial* de L_1 a L_2 ($g : L_1 \leqslant^p L_2$).

- Informalmente: L_1 es **más fácil** de **decidir** que L_2 (o L_2 es **más difícil** que L_1).

Reducibilidad en tiempo polinomial

Definición: Sean X_1, X_2 problemas de decisión. Diremos que X_1 **es reducible a** X_2 **en tiempo polinomial** $(X_1 \leqslant^p X_2)$ **sii** $L_{X_1} \leqslant^p L_{X_2}$.

- ⋆ Informalmente: resolver X_1 es **más fácil** que resolver X_2 (o bien resolver X_2 es **más difícil** que resolver X_1).

Sean $X_1 = (E_{X_1}, \theta_{X_1})$ y $X_2 = (E_{X_2}, \theta_{X_2})$ problemas de decisión. Una **reducibilidad en tiempo polinomial** de X_1 en X_2 es una función total $g : E_{X_1} \to E_{X_2}$, tal que:

- g es computable en tiempo polinomial.
- $\forall u \in E_{X_1}\ (\theta_{X_1}(u) = 1 \iff \theta_{X_2}(g(u)) = 1)$.

Reducibilidad en tiempo polinomial: propiedades

Proposición 1: Sean X_1, X_2, X_3 problemas de decisión. Se verifica:

(a) $X_1 \leqslant^p X_1$.

(b) Si $X_1 \leqslant^p X_2$ y $X_2 \leqslant^p X_3$ entonces $X_1 \leqslant^p X_3$.

Demostración:
(a): La aplicación identidad en X_1 es una r.t.p. de X_1 en sí mismo.

(b): Si g es una es una r.t.p de X_1 en X_2 y h es una r.t.p de X_2 en X_3, entonces $h \circ g$ es una es una r.t.p de X_1 en X_3.

■

Ahora bien: existen problemas de decisión X_1, X_2 tales que $X_1 \leqslant^p X_2$ y, en cambio, no se verifica que $X_2 \leqslant^p X_1$.

Definición: (*Equivalencia en tiempo polinomial*)
$X_1 \equiv^p X_2 \Longleftrightarrow X_1 \leqslant^p X_2 \wedge X_2 \leqslant^p X_1$.

Se verifica que: (a) $X_1 \equiv^p X_1$; (b) Si $X_1 \equiv^p X_2$ entonces $X_2 \equiv^p X_1$; y (c) si $X_1 \equiv^p X_2$ y $X_2 \equiv^p X_3$ entonces $X_1 \equiv^p X_3$.

Estabilidad de la clase P

Proposición 2: Sean X_1, X_2 problemas de decisión tales que $X_1 \leqslant^p X_2$ y $X_2 \in \mathbf{P}$. Entonces se tiene que $X_1 \in \mathbf{P}$.

Demostración:

Sea f una r.t.p. de X_1 en X_2 y sea $\mathcal{A}$ un algoritmo **determinista** de coste en tiempo polinomial que resuelve X_2.

Se considera el siguiente algoritmo **determinista** $\mathcal{B}$:

`Entrada:` $u \in E_{X_1}$.

`Hallar` $f(u) \in E_{X_2}$

`Ejecutar el algoritmo` $\mathcal{A}$ `con entrada` $f(u) \in E_{X_2}$

Coste en tiempo de $\mathcal{B}$ = coste en tiempo $\mathcal{A}$ + coste en tiempo de f (polinomial).

Además, el algoritmo $\mathcal{B}$ resuelve el problema X_1 ya que dado $u \in E_{X_1}$ se tiene que:

$$\theta_{X_1}(u) = 1 \overset{f \text{ r.t.p.}}{\Longleftrightarrow} \theta_{X_2}(f(u)) = 1 \overset{\mathcal{A} \text{ resuelve } X_2}{\Longleftrightarrow} \mathcal{A}(f(u)) = 1 \overset{\text{def. } \mathcal{B}}{\Longleftrightarrow} \mathcal{B}(u) = 1$$

■

Es decir: si un problema X pertenece a $\mathbf{P}$, entonces todo problema **más fácil** que X también pertenece a $\mathbf{P}$.

Estabilidad de la clase NP

Proposición 3: Sean X_1, X_2 problemas de decisión tales que $X_1 \leqslant^p X_2$ y $X_2 \in$ **NP**. Entonces se tiene que $X_1 \in$ **NP**.

Demostración:

Sea f una r.t.p. de X_1 en X_2 y sea $\mathcal{A}$ un algoritmo **no determinista** de coste en tiempo polinomial que resuelve X_2.

Se considera el siguiente algoritmo **no determinista** $\mathcal{B}$:

```
Entrada:  u ∈ E_{X_1} .
  Hallar f(u) ∈ E_{X_2}
  Para cada computación C de A(f(u)) hacer
    si C devuelve sí entonces
      devolver sí
    si no
      devolver no
```

Coste en tiempo de $\mathcal{B}$ = coste en tiempo $\mathcal{A}$ + coste en tiempo de f (polinomial).

Además, el algoritmo $\mathcal{B}$ resuelve el problema X_1 ya que dado $u \in E_{X_1}$ se tiene que:

$$\theta_{X_1}(u) = 1 \overset{f \text{ r.t.p.}}{\Longleftrightarrow} \theta_{X_2}(f(u)) = 1 \overset{\mathcal{A} \text{ resuelve } X_2}{\Longleftrightarrow} \text{existe alguna computación de aceptación en } \mathcal{A}(f(u))$$
$$\overset{\text{def. } \mathcal{B}}{\Longleftrightarrow} \text{existe alguna computación de aceptación en } \mathcal{B}(u)$$

■

Es decir: si un problema X pertenece a **NP**, entonces todo problema **más fácil** que X también pertenece a **NP**.

Problemas NP-completos

Definición: *Un problema de decisión X es* **NP-duro sii** *para cada problema $X' \in$* **NP** *se tiene que $X' \leqslant^p X$.*

* Un problema **NP-duro** es más difícil que cualquier problema de la clase **NP**.

Proposición 4: *Si X_1 es* **NP***-duro y $X_1 \leqslant^p X_2$, entonces X_2 es* **NP***-duro.*

Demostración: Para probar que X_2 es **NP**-duro, sea $Y \in$ **NP** y veamos que $Y \leqslant^p X_2$.

En efecto:

$$\left.\begin{array}{l} X_1 \text{ NP-duro} \\ Y \in \text{NP} \end{array}\right\} \Rightarrow Y \leqslant^p X_1 \left.\begin{array}{l} \\ \\ X_1 \leqslant^p X_2 \end{array}\right\} \overset{\text{Prop.1 (b)}}{\Longrightarrow} Y \leqslant^p X_2$$

■

Definición: *Un problema de decisión X es* **NP-completo sii** *$X \in$* **NP** *y X es* **NP***-duro.*

Los problemas **NP**-completos son los más difíciles de la clase **NP**:

* Para probar que **P** $\neq$ **NP**, los problemas **NP**-completos son los testigos ideales.

Problemas NP-completos

Proposición 5: *Si* X_1, X_2 *son problemas* **NP***–completos, entonces* $X_1 \equiv^p X_2$.

Demostración:

$$\left.\begin{array}{l} \left.\begin{array}{l} X_1 \text{ } \mathbf{NP}\text{-completo} \Rightarrow X_1 \in \mathbf{NP} \\ X_2 \text{ } \mathbf{NP}\text{-completo} \Rightarrow X_2 \text{ } \mathbf{NP}\text{-duro} \end{array}\right\} \Rightarrow X_1 \leqslant^p X_2 \\ \left.\begin{array}{l} X_1 \text{ } \mathbf{NP}\text{-completo} \Rightarrow X_1 \text{ } \mathbf{NP}\text{-duro} \\ X_2 \text{ } \mathbf{NP}\text{-completo} \Rightarrow X_2 \in \mathbf{NP} \end{array}\right\} \Rightarrow X_2 \leqslant^p X_1 \end{array}\right\} \Rightarrow X_1 \equiv^p X_2$$

■

Generación de problemas NP-completos

Teorema 1: *Sea X un problema de decisión tal que:*

- $X \in$ **NP**.
- *Existe* X' (X' *es* **NP***–completo* $\wedge\ X' \leqslant^p X$).

Entonces X es un problema **NP***–completo*.

Demostración:

Basta probar que el problema X es **NP**-duro. Se verifica:

$$\left.\begin{array}{r} X' \text{ es } \mathbf{NP}\text{-completo} \Rightarrow X' \text{ es } \mathbf{NP}\text{-duro} \\ X' \leqslant^p X \end{array}\right\} \stackrel{\text{Prop. } 4}{\Longrightarrow} X \text{ es } \mathbf{NP}\text{-duro}$$

■

Nacimiento de la teoría de la Complejidad Computacional

Teorema de Cook: SAT es un problema **NP**–completo (1971).

Proposición 6: *Son equivalentes:*

(a) $\mathbf{P} = \mathbf{NP}$.

(b) *Existe* **UN** *problema* **NP***–duro que pertenece a la clase* **P**.

Demostración:

(a) ⇒ (b): Supongamos que $\mathbf{P} = \mathbf{NP}$. Teniendo presente que SAT es un problema **NP**–completo, se deduce que $\mathtt{SAT} \in \mathbf{NP}$ $(= \mathbf{P})$ y, además, es **NP**–duro.

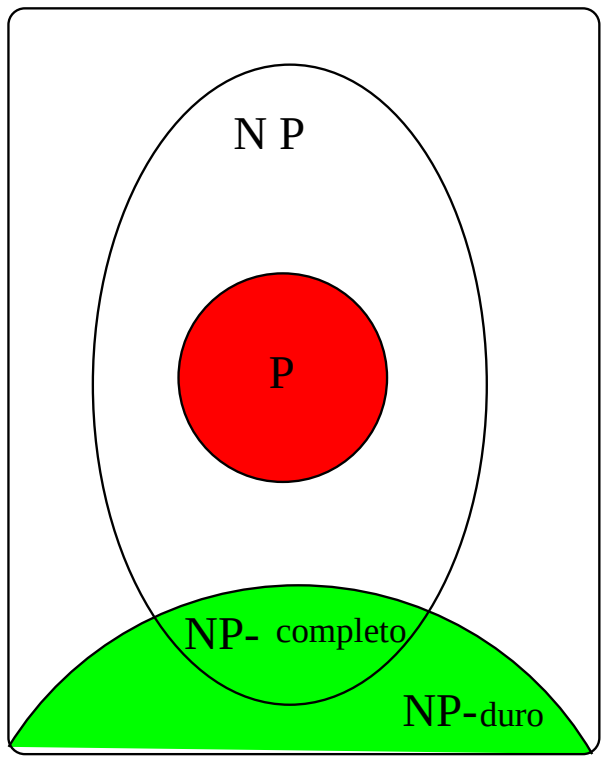
(b) ⇒ (a): Supongamos que X es un problema **NP**–duro tal que $X \in \mathbf{P}$. Para establecer que $\mathbf{NP} \subseteq \mathbf{P}$ hemos de ver que si $Y \in \mathbf{NP}$ entonces $Y \in \mathbf{P}$. Veámoslo:

$$\left.\begin{array}{r} X \text{ es } \mathbf{NP}\text{-duro} \\ Y \in \mathbf{NP} \end{array}\right\} \Longrightarrow Y \leqslant^p X \left.\begin{array}{r} \\ X \in \mathbf{P} \end{array}\right\} \overset{\text{Prop. 2}}{\Longrightarrow} Y \in \mathbf{P}$$

Del resultado de Cook y de la Proposición 6, se deduce que

$$\mathbf{P} = \mathbf{NP} \iff \mathtt{SAT} \in \mathbf{P}$$

La búsqueda de **un** adecuado problema **NP**–completo puede responder en sentido afirmativo o en sentido negativo a la conjetura $\mathbf{P} \neq \mathbf{NP}$.



El problema SAT

Definición: El **lenguaje de la Lógica Proposicional** consta de:

- Conjunto numerable, VP, de variables proposicionales.
- Conectivas lógicas: "$\neg$" (negación) y "$\vee$" (disyunción).
- Símbolos auxiliares: "(" y ")".

Definición: El conjunto $PForm$ de las **fórmulas proposicionales** es el menor conjunto, Γ, que contiene a VP y

- Si $P \in \Gamma$, entonces $\neg P \in \Gamma$.
- Si $P, Q \in \Gamma$, entonces $(P \vee Q) \in \Gamma$.

A partir de $\neg$ y $\vee$ se definen las conectivas lógicas $\wedge$, $\rightarrow$ y $\leftrightarrow$.

Usualmente, notaremos

- $\neg P$ así $\overline{P}$.
- $P \vee Q$ así $P + Q$.
- $P \wedge Q$ así $P \cdot Q$.

Definición: Un **literal** es una variable proposicional o la negación de una variable proposicional.

Definición: Una **cláusula** es la disyunción de un número finito de literales.

Definición: Una fórmula proposicional está en **forma normal conjuntiva** (**FNC**) si es la conjunción de un número finito de cláusulas.

Toda fórmula proposicional en FNC es la conjunción de una disyunción de literales (las supondremos en **forma "simplificada"**: las cláusulas y las variables no aparecerán repetidas ni serán complementarias).

Definición: Una **valoración** (**asignación**) **de verdad** es una aplicación de VP en $\{0, 1\}$.

- ⋆ Toda valoración de verdad se extiende de manera natural a una aplicación de $PForm$ en $\{0, 1\}$ (a través de las **tablas de verdad**).

Valoraciones de verdad **relevantes** para una fórmula.

- ⋆ Una valoración relevante para una fórmula es una aplicación del conjunto de sus variables en $\{0, 1\}$.
- ⋆ Si φ tiene n variables, entonces el número total de valoraciones relevantes para φ es 2^n.

Definición: Dos fórmulas proposicionales son **semánticamente equivalentes sii** cualquier valoración le asigna a ambas el mismo valor.

- Toda fórmula proposicional tiene una fórmula semánticamente equivalente en FNC (y en forma simplificada).

Definición: Una fórmula proposicional, φ, es **satisfactible sii** existe, al menos, una valoración, σ, tal que $\sigma(\varphi) = 1$.

Problema SAT: Dada una fórmula proposicional en FNC (y en forma simplificada), **determinar si es satisfactible**.

Proposición 7: *El problema* SAT *pertenece a la clase* **NP**.

Demostración: Consideremos el siguiente algoritmo $\mathcal{A}$ no determinista:

Entrada: $\varphi \in E_{\mathtt{SAT}}$ (sea $Var(\varphi) = \{x_1, \ldots, x_n\}$).

$\sigma \leftarrow \emptyset$

para cada $i = 1$ hasta n hacer
  Elegir e_1, e_2
    $e_1 : \sigma \leftarrow \sigma \cup \{(x_i, 0)\}$
    $e_2 : \sigma \leftarrow \sigma \cup \{(x_i, 1)\}$

} **Fase no determinista**

hallar $\sigma(\varphi)$
si $\sigma(\varphi) = 1$ entonces
  devolver sí
si no
  devolver no

} **Fase determinista**

Hallemos el coste en tiempo del algoritmo $\mathcal{A}$.

* Tiempo de la fase no determinista: exactamente n pasos (en cada una de las computaciones).
* Tiempo de la fase determinista: el número total de literales de la fórmula.

El coste en tiempo del algoritmo $\mathcal{A}$ es lineal en el tamaño de la fórmula de entrada (número total de literales de la fórmula).

Veamos que el algoritmo $\mathcal{A}$ resuelve el problema SAT. Para ello, sea φ una fórmula de entrada del problema.

* Supongamos que φ es satisfactible. Sea σ_0 una valoración tal que $\sigma_0(\varphi) = 1$. Sea $\mathcal{C}$ la computación que en la fase no determinista selecciona, precisamente, σ_0. Entonces esa computación $\mathcal{C}$ devolverá **sí** y, por tanto, será de aceptación.
* Supongamos que φ **no** es satisfactible. Sea $\mathcal{C}$ cualquier computación y notemos σ_1 la valoración seleccionada en la fase no determinista. Entonces $\sigma_1(\varphi) = 0$ y, por tanto, la computación $\mathcal{C}$ **no** será de aceptación.

En consecuencia, el problema SAT pertenece a la clase **NP**.

■

Teorema 2 (S.A. Cook, 1971) **:** *El problema* SAT *es* **NP***–duro.*

Idea de la demostración:

Sea $X = (\Sigma_X, I_X, \theta_X)$ un problema de la clase **NP**. Veamos que $X \leqslant^p$ SAT.

Puesto que $X \in$ **NP**, existirá una MTND, M, que trabaja en tiempo polinomial y resuelve X; es decir, existirá $k \geqslant 1$ tal que:

- $\star$ Para cada $u \in \Sigma_X^*$, se tiene que $\theta_X(u) = 1$ **sii** existe alguna computación de $M(u)$ que devuelve **sí** en, a lo sumo, $|u|^k$ pasos.

Restricciones para M:

- $\Gamma = \Sigma_X$ y $F = \{q_y, q_n\}$.
- M tiene una sóla cinta infinita con primera casilla.
- La función de transición δ es total.
- Aceptar $\equiv q_y$; Parar y no aceptar $\equiv q_n$.
- En la primera casilla de la configuración inicial aparecerá el primer símbolo del dato de entrada.
- En la configuración de aceptación, el cabezal analiza/inspecciona la primera casilla y todas las casillas están en blanco.

Simularemos $|u|^k$ pasos de todas las computaciones de M sobre $u = u_1 \dots u_n$.

* En cada computación se analiza/inspecciona, a lo sumo, $1 + |u|^k$ casillas.

Consideremos las fórmulas:

* $A(i, q)$ $\equiv$ en el instante i, el estado de M es q.
* $B(i, j)$ $\equiv$ en el instante i, el cabezal analiza/inspecciona la casilla j.
* $C(i, j, s) \equiv$ en el instante i, la casilla j contiene el símbolo s.

En donde $0 \leqslant i \leqslant |u|^k$, $1 \leqslant j \leqslant 1 + |u|^k$, $q \in Q$ y $s \in \Sigma_X$.

* **Configuración inicial**:

$$A(0, q_0) \ \wedge \ B(0, 1) \ \wedge \bigwedge_{1 \leqslant j \leqslant |u|} C(0, j, u_j) \ \wedge \bigwedge_{|u|+1 \leqslant j \leqslant 1+|u|^k} C(0, j, B) \qquad (1)$$

* **Configuración de aceptación**:

$$A(|u|^k, q_y) \ \wedge \ B(|u|^k, 1) \ \wedge \bigwedge_{1 \leqslant j \leqslant 1+|u|^k} C(|u|^k, j, B) \qquad (2)$$

Requisitos de cualquier computación de $M(u)$:

(a) En cada instante, M está en un sólo estado:

$$\forall i\ (0 \leqslant i \leqslant |u|^k \rightarrow \bigvee_{q \in Q} A(i,q) \wedge \bigwedge_{\substack{p,q \in Q \\ p \neq q}} [(\neg A(i,p)) \vee (\neg A(i,q))]) \quad (3)$$

(b) En cada instante, cada casilla contiene un sólo símbolo:

$$\forall i \forall j\ (0 \leqslant i \leqslant |u|^k \wedge 1 \leqslant j \leqslant 1 + |u|^k \rightarrow \bigvee_{s \in \Sigma_X} C(i,j,s) \wedge \bigwedge_{\substack{s,t \in \Sigma_X \\ s \neq t}} [(\neg C(i,j,s)) \vee (\neg C(i,j,t))]) \quad (4)$$

(c) En cada instante, el cabezal analiza/inspecciona una única casilla:

$$\forall i\ (0 \leqslant i \leqslant |u|^k \rightarrow \bigvee_{1 \leqslant j \leqslant 1+|u|^k} B(i,j) \wedge \bigwedge_{\substack{1 \leqslant j,j' \leqslant 1+|u|^k \\ j \neq j'}} [(\neg B(i,j)) \vee (\neg B(i,j'))]) \quad (5)$$

Codificación de la función de transición:

* La configuración en el paso $i+1$ resulta de la del paso i aplicando δ:

$$A(i,q) \wedge B(i,j) \wedge C(i,j,s) \rightarrow \bigvee_{((q,s),(q',s',r)) \in G(\delta)} A(i+1,q') \wedge B(i+1,j+r)) \wedge C(i+1,j,s') \quad (6)$$

  En donde $G(\delta)$ es la gráfica de la función de transición y $r \in \{0,+1,-1\}$ (equivalente a $\{$ **S**, **R**, **L**$\}$).

* Si una casilla no se analiza/inspecciona en el instante i entonces conserva su símbolo en el instante siguiente $i+1$: $C(i,j,s) \wedge \neg B(i,j) \rightarrow C(i+1,j,s)$ (7)

Se considera la fórmula

$$\varphi_u \equiv FNC[(1) \wedge (2) \wedge (3) \wedge (4) \wedge (5) \wedge \bigwedge_{\substack{0 \leqslant i \leqslant |u|^k \\ 1 \leqslant j \leqslant 1+|u|^k}} ((6) \wedge (7))]$$

Para cada $u \in \Sigma_X^*$ la fórmula φ_u

- Está en **FNC**.
- Su longitud es polinomial en $|u|$.
- Puede ser construida a partir de u en tiempo polinomial.

La aplicación $F : \Sigma_X^* \to E_{\mathtt{SAT}}$ definida por $F(u) = \varphi_u$ es total y computable en tiempo polinomial.

Además, para cada $u \in \Sigma_X^*$ se tiene que: $\theta_X(u) = 1$ **sii** existe alguna computación de aceptación de $M(u)$; es decir, **sii** φ_u es satisfactible.

Luego $X \leqslant^p \mathtt{SAT}$ y, en consecuencia, el problema $\mathtt{SAT}$ es **NP**–duro.

■

Variantes del problema SAT

Problema SAT (k) con $k \geq 1$: **Dada una fórmula proposicional en FNC en la que cada cláusula posee, a lo sumo, k literales, determinar si es satisfactible.**

Proposición 8: $\forall k \geqslant 1$ $(\mathtt{SAT}(k) \leqslant^p \mathtt{SAT})$.

Demostración: Consideremos la aplicación $F : E_{\mathtt{SAT}(k)} \longrightarrow E_{\mathtt{SAT}}$ definida así: $F(\varphi) = \varphi$, para cada $\varphi \in E_{\mathtt{SAT}(k)}$. Obviamente, F es una reducibilidad en tiempo polinomial de $\mathtt{SAT}(k)$ en $\mathtt{SAT}$.

■

Corolario: $\forall k \geqslant 1$ $(\mathtt{SAT}(k) \in \mathbf{NP})$.

Demostración: Basta tener presente la estabilidad de la clase **NP** (proposición 3).

■

Proposición 9: $\forall k \geqslant 1$ $(\mathtt{SAT}(k) \leqslant^p \mathtt{SAT}(k+1))$.

Demostración: Consideremos la aplicación $F : E_{\mathtt{SAT}(k)} \longrightarrow E_{\mathtt{SAT}(k+1)}$ definida así: $F(\varphi) = \varphi$, para cada $\varphi \in E_{\mathtt{SAT}(k)}$. Obviamente, F es una aplicación "**bien definida**" y, además, es una reducibilidad en tiempo polinomial de $\mathtt{SAT}(k)$ en $\mathtt{SAT}(k+1)$.

■

Teorema 3: *El problema* $\texttt{SAT}(3)$ *es* **NP***–completo.*

Demostración: Basta probar que $\texttt{SAT} \leqslant^p \texttt{SAT}(3)$.

- Para ello, a cada cláusula c de $\varphi \in E_{\texttt{SAT}}$ se le asocia una fórmula $D_c \in E_{\texttt{SAT}(3)}$ definida como sigue:
  - Si c posee, a lo sumo, 3 literales, entonces $D_c = c$.
  - Si $c = l_1 + \cdots + l_n$ $(n \geqslant 4)$, entonces

$$D_c = (l_1 + \overline{z}_1) \cdot (l_2 + z_1 + \overline{z}_2) \cdots (l_n + z_{n-1} + \overline{z}_n) \cdot z_n$$

    en donde $z_1, \ldots, z_n$ son nuevas variables (distintas de las de c).

Se considera la aplicación $F : E_{\texttt{SAT}} \to E_{\texttt{SAT}(3)}$ definida así:

$$F(c_1 \cdots c_p) = D_{c_1} \cdots D_{c_p}$$

Veamos que F es computable en tiempo polinomial.

* El siguiente programa calcula F en tiempo polinomial:

```
Entrada: φ ≡ c_1 · ··· · c_p con c_i ≡ l_i^1 + ··· + l_i^{r_i}   (1 ⩽ i ⩽ p)
        para i ← 1 hasta p hacer
                si r_i ⩽ 3 entonces
                        D_{c_i} ← c_i
                si no
                        Considerar nuevas variables z_i^1, ..., z_i^{r_i}
                        D_{c_i} ← (l_i^1 + z̄_i^1) · z_i^{r_i}
                        para j ← 2 hasta r_i hacer
                                D_{c_i} ← D_{c_i} · (l_i^j + z_i^{j-1} · z̄_i^j)
        devolver D_{c_1} ... D_{c_p}
```

Coste en tiempo: $\theta(r_1 + \cdots + r_p)$.

Veamos que $\forall \varphi \in E_{\mathtt{SAT}}\ (\varphi \in L_{\mathtt{SAT}} \Longleftrightarrow F(\varphi) \in L_{\mathtt{SAT(3)}})$; es decir, para cada $\varphi \in E_{\mathtt{SAT}}$ se tiene que φ es satisfactible **sii** $F(\varphi)$ es satisfactible.

Lema 1: *Para cada cláusula c se tiene que c es satisfactible* **sii** *D_c es satisfactible.*

Demostración: Si c tiene, a lo sumo, tres literales, el resultado es obvio (ya que $c = D_c$). Sea c una cláusula tal que $c = l_1 + \ldots + l_n$ (con $n \geqslant 4$). Entonces, $D_c = (l_1 + \overline{z}_1) \cdot (l_2 + z_1 + \overline{z}_2) \cdots (l_n + z_{n-1} + \overline{z}_n) \cdot z_n$.

* Supongamos que c es satisfactible. Sea σ una valoración tal que $\sigma(c) = 1$ y consideremos $j_0 = \text{mín}\{j :\ \sigma(l_j) = 1\}$. Sea τ la valoración:

$$\begin{cases} \tau(l_j) = \sigma(l_j) & \text{si } 1 \leqslant j \leqslant n \\ \tau(z_j) = 0 & \text{si } 1 \leqslant j < j_0 \\ \tau(z_j) = 1 & \text{si } j_0 \leqslant j \leqslant n \\ \tau(x) = 1 & \text{para cualquier otra variable} \end{cases}$$

  Se tiene que $\tau(D_c) = 1$.

* Supongamos que D_c es satisfactible. Sea σ una valoración tal que $\sigma(D_c) = 1$, entonces se verifica que $\sigma(c) = 1$. En efecto, de lo contrario, $\sigma(l_1) = \cdots = \sigma(l_n) = 0$. En tal situación, se prueba (por inducción acotada) que $\sigma(z_1) = \cdots = \sigma(z_n) = 0$.

  Lo que es una contradicción (pues z_n es una cláusula de la fórmula D_c).

■

Lema 2: *Para cada* $\varphi \in E_{\mathtt{SAT}}$ *se tiene que* $\varphi \in L_{\mathtt{SAT}} \Longleftrightarrow F(\varphi) \in L_{\mathtt{SAT(3)}}$

Demostración: Sea $\varphi \equiv c_1 \cdot \cdots \cdot c_p$ con $c_i \equiv l_i^1 + \ldots + l_i^{r_i}$.

* Supongamos que φ es satisfactible. Sea σ una valoración tal que $\sigma(\varphi) = 1$. Entonces, para cada i, $1 \leqslant i \leqslant p$, $\sigma(c_i) = 1$. Luego, del lema 1 resulta que D_{c_i} es satisfactible por una valoración que "extiende" σ.

  Considerando, en su caso, las nuevas variables en D_{c_i} distintas todas ellas entre sí, para $1 \leq i \leq p$, existirá una extensión de σ que hace verdadera las fórmulas $D_{c_1}, \ldots, D_{c_p}$. Luego $F(\varphi)$ será satisfactible.

* Supongamos que $F(\varphi)$ es satisfactible. Sea σ una valoración tal que $\sigma(F(\varphi)) = 1$. Entonces $\sigma(D_{c_1}) = \ldots = \sigma(D_{c_p}) = 1$. De la prueba del lema 1 resulta que $\sigma(c_1) = \ldots = \sigma(c_p) = 1$. Luego, $\sigma(\varphi) = 1$.

■

En consecuencia: $\texttt{SAT} \leqslant^p \texttt{SAT}(3)$.

* Como $\texttt{SAT}$ es **NP**–completo y $\texttt{SAT}(3) \in \textbf{NP}$, del teorema de generación de problemas **NP**–completos se concluye que $\texttt{SAT}(3)$ es **NP**-completo.

■

Corolario: $\forall k \geqslant 3$ ($\texttt{SAT}(k)$ *es* **NP**-*completo*).

Demostración: Por inducción sobre k (para $k \geq 3$).

* Caso base: $k = 3$. Se deduce del teorema anterior.
* Paso inductivo: Sea $k \geq 3$ tal que $\texttt{SAT}(k)$ es **NP**-completo. Veamos que $\texttt{SAT}(k+1)$ también es **NP**-completo. En efecto:
  - Por una parte, del corolario de la proposición 8 resulta que $\texttt{SAT}(k+1) \in \textbf{NP}$.
  - Por otra parte, de la proposición 9 resulta que $\texttt{SAT}(k) \leqslant^p \texttt{SAT}(k+1)$.

  De estas relaciones y del teorema de generación de problemas **NP**-completos, se deduce que $\texttt{SAT}(k+1)$ es también **NP**-completo.

■

Problema k–SAT, para $k \geq 1$

Problema k-SAT con $k \geq 1$: **Dada una fórmula proposicional en FNC en la que cada cláusula posee, exactamente, k literales, determinar si es satisfactible.**

El problema k–SAT se denota, a veces, así: $\mathtt{SAT}^*(k)$.

Proposición 10: $\forall k \geqslant 1$ (k–SAT $\leqslant^p$ SAT).

Demostración: Consideremos la aplicación $F : E_{k-\mathtt{SAT}} \longrightarrow E_{\mathtt{SAT}}$ definida como sigue: $F(\varphi) = \varphi$, para cada $\varphi \in E_{k-\mathtt{SAT}}$. Obviamente, F es una reducibilidad en tiempo polinomial de k–SAT en SAT.

■

Corolario: $\forall k \geqslant 1$ (k–SAT $\in$ **NP**).

Demostración: Basta tener presente la estabilidad de la clase **NP** (proposición 3).

■

Proposición 11: $\forall k \geqslant 1$ (k–SAT $\leqslant^p$ $(k+1)$–SAT).

Demostración: Consideremos la aplicación $F : E_{k-\mathtt{SAT}} \longrightarrow E_{(k+1)-\mathtt{SAT}}$ definida como sigue: si $\varphi = c_1 \cdot \cdots \cdot c_p \in E_{k-\mathtt{SAT}}$ entonces

$$F(\varphi) = (c_1 + z_1) \cdot (c_1 + \overline{z}_1) \cdot \cdots \cdot (c_p + z_p) \cdot (c_p + \overline{z}_p)$$

siendo $z_1, \ldots, z_p$ nuevas variables (que no aparecen en las cláusulas $c_1, \ldots, c_p$).

Veamos que F es una reducibilidad en tiempo polinomial de k–SAT en $(k+1)$–SAT. En efecto:

- $F(\varphi) \in E_{(k+1)-\mathtt{SAT}}$, para cada $\varphi \in E_{k-\mathtt{SAT}}$.
- F es computable en tiempo polinomial.
- Si φ es satisfactible, existe una valoración σ tal que $\sigma(\varphi) = 1$. Luego $\sigma(c_1) = \ldots = \sigma(c_p) = 1$. Por tanto, $\sigma(F(\varphi)) = 1$
- Si $F(\varphi)$ es satisfactible, existe una valoración τ tal que $\tau(F(\varphi)) = 1$. Luego $\tau(c_1 + z_1) = \tau(c_1 + \overline{z}_1) = \ldots = \tau(c_p + z_p) = \tau(c_p + \overline{z}_p) = 1$ y, por tanto, $\tau(c_1) = \ldots = \tau(c_p) = 1$; es decir $\tau(\varphi) = 1$

En consecuencia, k–SAT $\leqslant^p$ $(k+1)$–SAT.

■

Teorema 4: *El problema* 3–SAT *es* **NP**–*completo.*

Demostración: Basta ver que $\mathtt{SAT}(3) \leqslant^p$ 3–SAT.

- Para cada cláusula c de $\varphi \in E_{\mathtt{SAT}(3)}$ se define la fórmula D_c como sigue:
  - Si $c = l_1$, entonces
    $D_c = (l_1 + z_1 + z_2) \cdot (l_1 + \overline{z}_1 + z_2) \cdot (l_1 + z_1 + \overline{z}_2) \cdot (l_1 + \overline{z}_1 + \overline{z}_2)$
    en donde z_1, z_2 son nuevas variables.
  - Si $c = l_1 + l_2$, entonces
    $$D_c = (l_1 + l_2 + t) \cdot (l_1 + l_2 + \overline{t})$$
    en donde t es una nueva variable.
  - Si c posee exactamente 3 literales, $D_c = c$.

Sea $F : E_{\mathtt{SAT}(3)} \to E_{3-\mathtt{SAT}}$ definida así:

$$F(c_1 \cdots c_p) = D_{c_1} \cdots D_{c_p}$$

Veamos que F es computable en tiempo polinomial.

* El siguiente programa calcula F en tiempo polinomial:

Entrada: $\varphi \equiv c_1 \cdots c_p$ con $c_i \equiv l_i^1 + \cdots + l_i^{r_i}$ $(1 \leqslant i \leqslant p,\ \ 1 \leqslant r_i \leqslant 3)$

para $i \leftarrow 1$ **hasta** p **hacer**

si $r_i = 1$ **entonces**

Considerar dos nuevas variables z_i^1, z_i^2

$D_{c_i} \leftarrow (l_i^1 + z_i^1 + z_i^2) \cdot (l_i^1 + \overline{z}_i^1 + z_i^2) \cdot (l_i^1 + z_i^1 + \overline{z}_i^2) \cdot (l_i^1 + \overline{z}_i^1 + \overline{z}_i^2)$

si $r_i = 2$ **entonces**

Considerar una nueva variable z_i

$D_{c_i} \leftarrow (l_i^1 + l_i^2 + z_i) \cdot (l_i^1 + l_i^2 + \overline{z}_i)$

si $r_i = 3$ **entonces**

$D_{c_i} \leftarrow c_i$

devolver $D_{c_1} \ldots D_{c_p}$

Coste en tiempo: $\theta(r_1 + \cdots + r_p)$.

Veamos que $\forall \varphi \in E_{\mathtt{SAT(3)}}$ $(\varphi \in L_{\mathtt{SAT(3)}} \iff F(\varphi) \in L_{3-\mathtt{SAT}})$; es decir, para cada $\varphi \in E_{\mathtt{SAT(3)}}$ se tiene que φ es satisfactible **sii** $F(\varphi)$ es satisfactible.

Lema 1: *Una cláusula c con, a lo sumo, tres literales es satisfactible* **sii** *D_c es satisfactible.*

Demostración: Si c posee exactamente tres literales entonces $c = D_c$. Luego el resultado es trivial. Supongamos, pues, que c posee, a los sumo, dos literales. Sea σ una valoración tal que $\sigma(c) = 1$ y sea τ una valoración que extiende a σ.

- Caso 1: Sea $c = l_1$. Entonces $\tau(l_1) = 1$. Luego

  $\tau(D_c) = \tau(l_1 + z_1 + z_2) \cdot \tau(l_1 + \overline{z}_1 + z_2) \cdot \tau(l_1 + z_1 + \overline{z}_2) \cdot \tau(l_1 + \overline{z}_1 + \overline{z}_2) = 1$

- Caso 2: Sea $c = l_1 + l_2$. Como $\tau(l_1 + l_2) = \sigma(l_1 + l_2) = 1$, se tiene que $\tau(D_c) = \tau(l_1 + l_2 + t) \cdot \tau(l_1 + l_2 + \overline{t}) = 1$.

Sea σ una valoración tal que $\sigma(D_c) = 1$.

- Caso 1: Sea $c = l_1$. Entonces $\sigma(l_1) = 1$, pues de lo contrario $\sigma(z_1) = \sigma(\overline{z}_1) = 1$.
- Caso 2: Sea $c = l_1 + l_2$. Si $\sigma(l_1) = \sigma(l_2) = 0$, entonces $\sigma(t) = \sigma(\overline{t}) = 1$.

■

Lema 2: *Para cada* $\varphi \in E_{\texttt{SAT(3)}}$ *se tiene que* $\varphi \in L_{\texttt{SAT(3)}} \Longleftrightarrow F(\varphi) \in L_{3-\texttt{SAT}}$.

Demostración: Sea $\varphi \equiv c_1 \cdots c_p \in E_{\texttt{SAT(3)}}$.

- Supongamos que φ es satisfactible. Sea σ una valoración tal que $\sigma(\varphi) = 1$. Entonces $\sigma(c_1) = \cdots = \sigma(c_p) = 1$. Si τ es una valoración que extiende a σ, entonces $\tau(D_{c_1}) = \cdots = \tau(D_{c_p}) = 1$.
- Supongamos que $F(\varphi)$ es satisfactible. Sea σ una valoración tal que $\sigma(F(\varphi)) = 1$. Entonces $\sigma(D_{c_1}) = \cdots = \sigma(D_{c_p}) = 1$. Razonando como en el lema 1, se tiene $\sigma(c_1) = \cdots = \sigma(c_p) = 1$. Por tanto, $\sigma(\varphi) = 1$.

■

En consecuencia, se tiene que $\texttt{SAT(3)} \leqslant^p$ 3–SAT.

Como SAT(3) es **NP**–completo y 3–SAT $\in$ **NP**, del teorema de generación de problemas **NP**–completos se concluye que 3–SAT es **NP**-completo.

■

Corolario: $\forall k \geqslant 3$ (k–SAT es **NP**-completo).

Demostración: Se prueba por inducción sobre k.

* Caso base: $k = 3$. Se deduce del teorema anterior.
* Paso inductivo: Sea $k \geq 3$ tal que k-SAT es **NP**-completo. Veamos que $(k+1)$-SAT también es **NP**-completo. En efecto:
  * Por una parte, del corolario de la proposición 10 resulta que $(k+1)-$SAT $\in$ **NP**.
  * Por otra, de la proposición 11 resulta que k-SAT $\leqslant^p$ $(k+1)-$SAT.

  De estas relaciones y del teorema de generación de problemas **NP**-completos, se deduce que $(k+1)$-SAT es también **NP**-completo.

■

Proposición 12: $2\text{–}\mathtt{SAT} \in \mathbf{P}$.

La demostración de este resultado se propondrá como un trabajo para la evaluación de la asignatura.

* **Corolario:** $1\text{–}\mathtt{SAT} \in \mathbf{P}$.

Proposición 13: $\mathtt{SAT}(2) \leqslant^p 2\text{–}\mathtt{SAT}$.

Demostración: Es similar (pero más fácil) a la prueba realizada en el teorema 4 de que $\mathtt{SAT}(3) \leqslant^p 3\text{–}\mathtt{SAT}$.

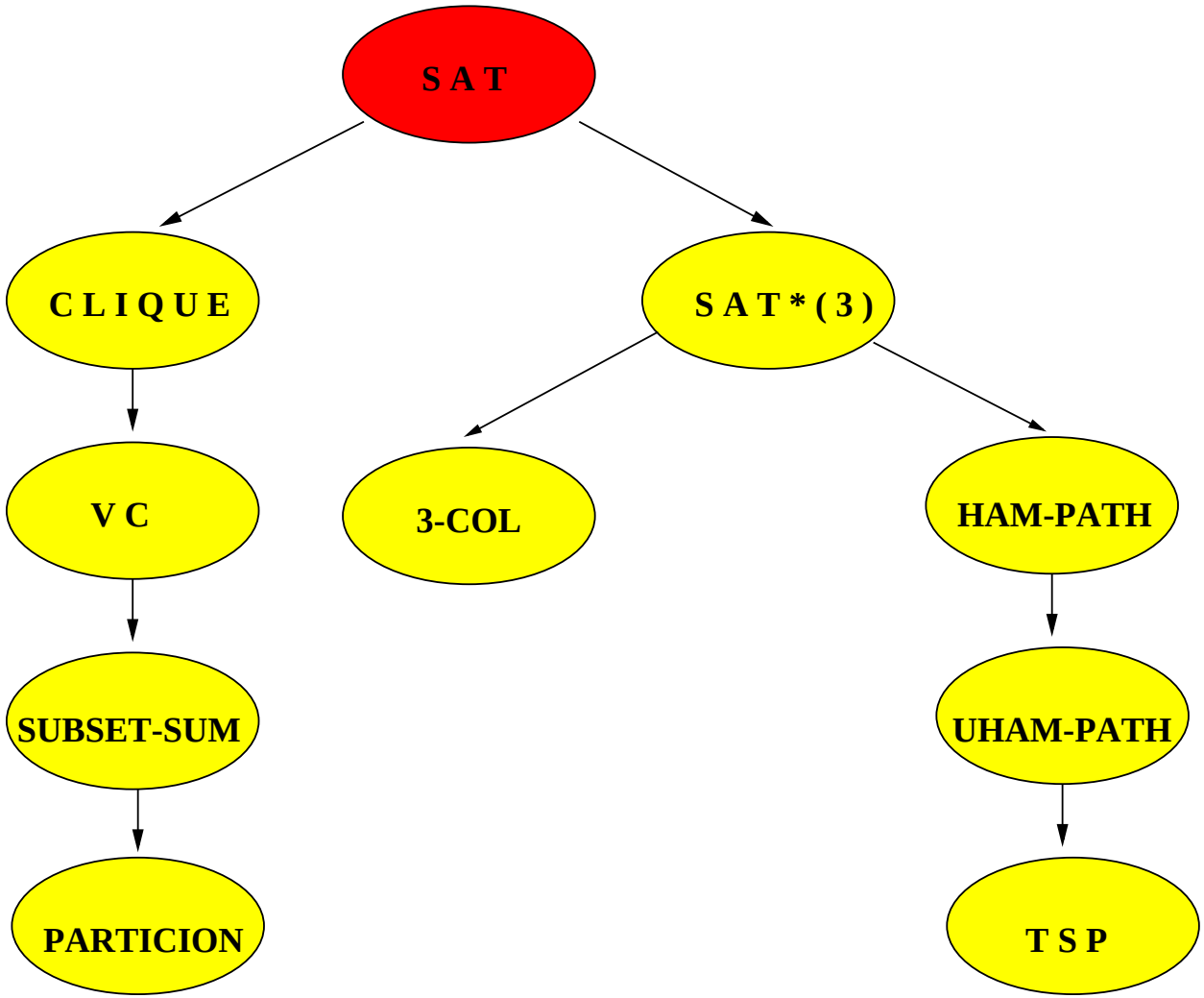
* **Corolario:** $\mathtt{SAT}(2) \in \mathbf{P}$ *y* $\mathtt{SAT}(1) \in \mathbf{P}$.

Problemas relevantes NP-completos

* Necesidad de obtener problemas **NP**–completos para abordar/atacar la conjetura $\mathbf{P} \stackrel{?}{\neq} \mathbf{NP}$.

* Nuevos problemas **NP**–completos (a partir del teorema de Cook).

  * Teorema de generación de problemas **NP**–completos.


SAT
CLIQUE
SAT*(3)
VC
3-COL
HAM-PATH
SUBSET-SUM
UHAM-PATH
PARTICION
TSP

El problema CLIQUE

Sea $G = (V, E)$ un grafo no dirigido. Diremos que un conjunto $V_1 \subseteq V$ es un **clique** (o **subgrafo completo**) de G **sii** para cada $u, w \in V_1$, con $u \neq w$, se tiene que $\{u, w\} \in E$.

Es decir: en un clique V_1 de un grafo no dirigido G, dos nodos distintos cualesquiera de V_1 siempre están conectados por una arista de G.

Problema CLIQUE: **Dado un grafo no dirigido $G = (V, E)$ y un número natural $k \in \mathbf{N}$, $1 \leq k \leq |V|$, determinar si existe un clique de G de tamaño k.**

NP-completitud del problema CLIQUE

Proposición 14: CLIQUE $\in$ **NP**.

Demostración: Consideremos el siguiente algoritmo $\mathcal{A}$ no determinista:

Entrada: (G, k), siendo $G = (V, E)$ un grafo no dirigido con $V = \{x_1, \ldots, x_n\}$ y $k \in \mathbf{N}, 1 \leq k \leq |V|$.

$V_1 \leftarrow \emptyset$

para cada $i = 1$ hasta n hacer
  Elegir e_1, e_2
    $e_1 : V_1 \leftarrow V_1$
    $e_2 : V_1 \leftarrow V_1 \cup \{x_i\}; k \leftarrow k - 1$

} **Fase no determinista**

si $k = 0$ y V_1 es un clique de G entonces
  devolver sí
si no
  devolver no

} **Fase determinista**

Hallemos el coste en tiempo del algoritmo $\mathcal{A}$.

* Tiempo de la fase no determinista: exactamente n pasos (en cada una de las computaciones).
* Tiempo de la fase determinista: cuadrático en el número de nodos del grafo.

El coste en tiempo del algoritmo $\mathcal{A}$ es cuadrático en el número de nodos del grafo de entrada; se decir, lineal en el "tamaño" de dicho grafo.

Veamos que el algoritmo $\mathcal{A}$ resuelve el problema `CLIQUE`. Para ello, sea (G, k) una instancia del problema.

* Supongamos que G posee un clique V_1 de tamaño k. Sea $\mathcal{C}$ la computación que en la fase no determinista selecciona, precisamente, el subconjunto V_1. Entonces esa computación $\mathcal{C}$ devolverá **sí** y, por tanto, será de aceptación.
* Supongamos que G **no** posee un clique de tamaño k. Sea $\mathcal{C}$ cualquier computación y notemos V_1' el subconjunto de V, de tamaño k, seleccionado en la fase no determinista. Entonces V_1' no será un clique de G y, por tanto, la computación $\mathcal{C}$ **no** será de aceptación.

En consecuencia, el problema `CLIQUE` pertenece a la clase **NP**.

■

Teorema 5: *El problema* CLIQUE *es* **NP***–completo*.

Demostración: Veamos que $\texttt{SAT} \leqslant^{\mathrm{p}} \texttt{CLIQUE}$.

Sea $\varphi \in E_{\texttt{SAT}}$, con $\varphi \equiv c_1 \cdots c_p$ y $c_i = l_i^1 + \ldots + l_i^{r_i}$.

Construimos el grafo $G_\varphi = (V_\varphi, E_\varphi)$ como sigue:

$$V_\varphi = \{(i,1), \ldots, (i,r_i) : \ 1 \leqslant i \leqslant p\}$$
$$\{(i,j),(k,q)\} \in E_\varphi \iff i \neq k \wedge l_i^j \neq \bar{l}_k^q \wedge 1 \leqslant i,k \leqslant p \wedge 1 \leqslant j \leqslant r_i \wedge 1 \leqslant q \leqslant r_k$$

Consideramos la aplicación $F : E_{\texttt{SAT}} \to E_{\texttt{CLIQUE}}$ definida como sigue:
$F(\varphi) = (G_\varphi, p)$.

Vamos a probar que F es una reducibilidad en tiempo polinomial del problema SAT en el problema CLIQUE (con lo cual, del teorema de generación de problemas **NP**-completos, se deducirá la **NP**-completitud del problema CLIQUE).

* F es total computable en tiempo polinomial.

$$
\begin{array}{l}
\texttt{Entrada:} \quad \varphi = (l_1^1 + \cdots + l_1^{r_1}) \cdots (l_p^1 + \cdots l_p^{r_p}) \\
\quad \texttt{para } \alpha \leftarrow 1 \texttt{ hasta } p \texttt{ hacer} \\
\qquad \texttt{para } \beta \leftarrow 1 \texttt{ hasta } p \texttt{ hacer} \\
\qquad\quad \texttt{si } \alpha \neq \beta \texttt{ entonces} \\
\qquad\qquad \texttt{para } \gamma \leftarrow 1 \texttt{ hasta } r_\alpha \texttt{ hacer} \\
\qquad\qquad\quad \texttt{para } \delta \leftarrow 1 \texttt{ hasta } r_\beta \texttt{ hacer} \\
\qquad\qquad\qquad \texttt{si } l_\alpha^\gamma \neq \bar{l}_\beta^\delta \texttt{ entonces} \\
\qquad\qquad\qquad\quad A[i,j] \leftarrow 1
\end{array}
$$

En donde:

- A es la matriz de adyacencia del grafo G_φ.
- $i = r_1 + \cdots + r_{\alpha-1} + \gamma$ es la posición que ocupa l_α^γ en φ.
- $j = r_1 + \cdots + r_{\beta-1} + \delta$ es la posición que ocupa l_β^δ en φ.

Obviamente, el algoritmo es de coste en tiempo polinomial.

* Veamos que la fórmula $\varphi \equiv c_1 \cdot \cdots \cdot c_p$, con $c_i \equiv l_i^1 + \cdots + l_i^{r_i}$ $(1 \leqslant i \leqslant p)$, es satisfactible si y solo si el grafo G_φ posee un clique de tamaño p.

  * Supongamos que la fórmula φ es satisfactible. Sea σ una valoración tal que $\sigma(\varphi) = 1$.
    Para cada $i, 1 \leq i \leq p$, sea $j_i = \text{mín}\,\{k \in N \mid \sigma(l_i^k) = 1\}$. Consideremos el conjunto $V_1 = \{l_1^{j_1}, \ldots, l_p^{j_p}\}$. Entonces $\sigma(l_1^{j_1}) = \ldots = \sigma(l_p^{j_p}) = 1$ y, por tanto, de acuerdo con la definición de G_φ resulta que V_1 es un clique de dicho grafo de tamaño p.

  * Supongamos que el grafo G_φ posee un clique, V_1', de tamaño p.
    Entonces, el conjunto V_1' ha de ser del tipo $V_1' = \{l_1', \ldots, l_p'\}$, en donde l_i' será un literal de la cláusula c_i de φ.

    Consideremos la valoración τ definida como sigue: $\tau(l_i') = 1$ y, para las restantes variables. el valor de τ es 0. Entonces, τ está bien definida ya que en V_1' no puede aparecer un literal y su negación.

    De acuerdo con la definición anterior $\tau(c_i) = 1$, para cada cláusula c_i ya que, al menos, uno de los literales de c_i es verdadero por τ (concretamente, el literal l_i').

    Por tanto, $\tau(\varphi) = 1$; es decir, la fórmula φ es satisfactible.

■

El problema VC del recubrimiento de vértices

Sea $G = (V, E)$ un un grafo no dirigido. Diremos que un conjunto $V_1 \subseteq V$ es un **recubrimiento de vértices** (r.v.) de G **sii** para cada arista de G, al menos uno de los extremos de la arista pertenece a V_1.

Problema VC: Dado un grafo no dirigido $G = (V, E)$ y un número natural $k \in \mathbf{N}$, $1 \leq k \leq |V|$, determinar si existe un recubrimiento de vértices de G de tamaño k.

NP-completitud del problema VC

Proposición 15: $\texttt{VC} \in \mathbf{NP}$.

Demostración: Consideremos el siguiente algoritmo $\mathcal{A}$ no determinista:

Entrada: (G, k), siendo $G = (V, E)$ un grafo no dirigido con $V = \{x_1, \ldots, x_n\}$ y $k \in \mathbf{N}, 1 \leq k \leq |V|$.

$V_1 \leftarrow \emptyset$

para cada $i = 1$ hasta n hacer
  Elegir e_1, e_2
    $e_1 : V_1 \leftarrow V_1$
    $e_2 : V_1 \leftarrow V_1 \cup \{x_i\}; k \leftarrow k - 1$

} **Fase no determinista**

si $k = 0$ y V_1 es un r.v. de G entonces
  devolver sí
si no
  devolver no

} **Fase determinista**

Hallemos el coste en tiempo del algoritmo $\mathcal{A}$.

* Tiempo de la fase no determinista: exactamente n pasos (en cada una de las computaciones).
* Tiempo de la fase determinista: cuadrático en el número de nodos del grafo.

El coste en tiempo del algoritmo $\mathcal{A}$ es cuadrático en el número de nodos del grafo de entrada; se decir, lineal en el tamaño de dicho grafo.

Veamos que el algoritmo $\mathcal{A}$ resuelve el problema VC. Para ello, sea (G, k) una instancia del problema.

* Supongamos que G posee un r.v. V_1 de tamaño k. Sea $\mathcal{C}$ la computación que en la fase no determinista selecciona, precisamente, el subconjunto V_1. Entonces esa computación $\mathcal{C}$ devolverá **sí** y, por tanto, será de aceptación.
* Supongamos que G **no** posee un r.v. de tamaño k. Sea $\mathcal{C}$ cualquier computación y notemos V_1' el subconjunto de V, de tamaño k, seleccionado en la fase no determinista. Entonces V_1' no será un r.v. de G y, por tanto, la computación $\mathcal{C}$ **no** será de aceptación.

En consecuencia, el problema VC pertenece a la clase **NP**.

■

Teorema 6: *El problema* VC *es* **NP***–completo.*

Demostración: Veamos que CLIQUE $\leq^p$ VC.

Dado el grafo G, definimos el grafo complementario, $\overline{G} = (V; \overline{E})$, así:

$$\overline{E} = \{\{u, v\} : \ u \in V, v \in V, u \neq v, \{u, v\} \notin E\}$$

Sea $F : E_{\texttt{CLIQUE}} \to E_{\texttt{VC}}$ definida así: $F(G, k) = (\overline{G}, |G| - k)$

Vamos a probar que F es una reducibilidad en tiempo polinomial del problema CLIQUE en el problema VC (con lo cual, del teorema de generación de problemas **NP**-completos, se deducirá la **NP**-completitud del problema VC).

- F es total computable en tiempo polinomial.

  Entrada: (G, k), siendo $G = (V, E)$ un grafo no dirigido con $V = \{x_1, \ldots, x_n\}$ y $k \in \mathbf{N}, 1 \leq k \leq |V|$.

  $A[i,j] \leftarrow$ matriz de adyacencia de G

  para $i = 1$ hasta n hacer

    para $j = 1$ hasta n hacer

      si $A[i,j] = 0$ entonces $B[i,j] \leftarrow 1$

      si no

        $B[i,j] \leftarrow 0$

Obviamente, este algoritmo determinista calcula la función F y, además, es de coste en tiempo polinomial.

- Veamos que el grafo $G = (V, E)$ posee un clique de tamaño k **sii** el grafo complementario $\overline{G}$ posee un r.v. de tamaño $|G| - k$.
  - Supongamos que el grafo G posee un clique, V_1, de tamaño k. Veamos que $V_2 = V \setminus V_1$ es un r.v. de $\overline{G}$ (cuyo tamaño es, obviamente, $|V| - k$).

    En efecto: si $\{u, w\}$ es una arista de $\overline{G}$ entonces $u \in V_2$ ó $w \in V_2$ ya que, de lo contrario $u \notin V_2 \wedge w \notin V_2$ y, por tanto, se tendría que $u \in V_1$ y $w \in V_1$. Teniendo presente que $\{u, w\} \in \overline{E} \Rightarrow \{u, w\} \notin E$, resultaría que V_1 **no** sería un clique de G.
  - Supongamos que el grafo $\overline{G}$ posee un r.v., V_1', de tamaño $|V| - k$. Veamos que $V_2' = V \setminus V_1'$ es un clique de G (cuyo tamaño es, obviamente, k).

    En efecto: sean $u, w \in V_2'$ tales que $u \neq w$. Entonces ha de verificarse que $\{u, w\} \in E$ ya que, de lo contrario $\{u, w\} \in \overline{E} \wedge u \in V_2' \wedge w \in V_2'$; es decir, $\{u, w\} \in \overline{E} \wedge u \notin V_1' \wedge w \notin V_1'$. Lo que contradice que V_1' sea un r.v. de $\overline{G}$.

■

El problema SUBSET–SUM

Una función peso sobre un conjunto finito no vacío A es una aplicación w de $\mathbf{P}(A)$ en $\mathbf{N}$ tal que para cada $B \subseteq A$ se tiene que $w(B) = \sum_{x \in B} w(\{x\})$.

Para abreviar, notaremos $w(x)$ en lugar de $w(\{x\})$.

- *Dado un conjunto finito no vacío, A, una función peso, w, sobre A, y $k \in \mathbf{N}$, determinar si existe $B \subseteq A$ tal que $w(B) = k$.*

Proposición 1: SUBSET–SUM $\in$ NP.

Teorema 2: *El problema* **SUBSET–SUM** *es* **NP***–completo*.

Demostración: Veamos que **VC** $\leq^p$ **SUBSET–SUM**.

Sea $(G = (V, E), k) \in E_{\mathbf{VC}}$, con $V = \{0, \ldots, n-1\}$ y $E = \{e_0, \ldots, e_{p-1}\}$.

Sea $B = (b_{ij})$ la **matriz de incidencia** de G.

$$b_{ij} = \begin{cases} 1 & \text{si } e_j \text{ es incidente con el nodo } i \\ 0 & \text{e.c.o.c} \end{cases}$$

Sea $A = \{x_0, \ldots, x_{n-1}\} \cup \{y_0, \ldots, y_{p-1}\}$, en donde

$$x_i = (1, \underbrace{b_{i(p-1)}, b_{i(p-2)}, \ldots, b_{i0}}_{(p-1-j)}) \ (0 \leqslant i \leqslant n-1)$$

$$y_j = (0, \underbrace{0, 0, \ldots, 0, \quad 1 \quad , 0, \ldots, 0}) \ (0 \leqslant j \leqslant p-1)$$

x_0	$=$	$\mathbf{1}$	$b_{0(p-1)}$	$b_{0(p-2)}$	$b_{0(p-3)}$	$\ldots$	b_{02}	b_{01}	b_{00}
x_1	$=$	$\mathbf{1}$	$b_{1(p-1)}$	$b_{1(p-2)}$	$b_{1(p-3)}$	$\ldots$	b_{12}	b_{11}	b_{10}
x_2	$=$	$\mathbf{1}$	$b_{2(p-1)}$	$b_{2(p-2)}$	$b_{2(p-3)}$	$\ldots$	b_{22}	b_{21}	b_{20}
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
x_{n-1}	$=$	$\mathbf{1}$	$b_{(n-1)(p-1)}$	$b_{(n-1)(p-2)}$	$b_{(n-1)(p-3)}$	$\ldots$	$b_{(n-1)2}$	$b_{(n-1)1}$	$b_{(n-1)0}$
y_0	$=$	$\mathbf{0}$	0	0	0	$\ldots$	0	0	1
y_1	$=$	$\mathbf{0}$	0	0	0	$\ldots$	0	1	0
y_2	$=$	$\mathbf{0}$	0	0	0	$\ldots$	1	0	0
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$
y_{p-3}	$=$	$\mathbf{0}$	0	0	1	$\ldots$	0	0	0
y_{p-2}	$=$	$\mathbf{0}$	0	1	0	$\ldots$	0	0	0
y_{p-1}	$=$	$\mathbf{0}$	1	0	0	$\ldots$	0	0	0
		4^p	4^{p-1}	4^{p-2}	4^{p-3}	$\ldots$	4^2	4^1	4^0

Sea $w : A \to \mathbf{N}$ definida así:

- $w(z)$ es el número cuyo desarrollo en base 4 está representado por la $(p+1)$–tupla, z.

De la definición anterior resulta que

- $w(x_i) = 4^p + \sum_{j=0}^{p-1} b_{ij} \cdot 4^j$.
- $w(y_j) = 4^j$.

Sea $k' = k \cdot 4^p + \sum_{j=0}^{p-1} 2 \cdot 4^j \equiv k222\ldots222\ {}_{(4)}$.

Consideremos $F : E_{\mathbf{VC}} \to E_{\mathbf{SUBSET-SUM}}$ definida así: $F(G, k) = (A, w, k')$.

- F es total computable en tiempo polinomial.
- **Aserto:** *G tiene un recubrimiento de vértices de tamaño k sii existe $A' \subseteq A$ tal que $w(A') = k'$.*

  Prueba del aserto:

  - Sea $V' = \{i_1, \ldots, i_k\}$ un recubrimiento de vértices de G de tamaño k. Sea

    $A' = \{x_{i_1}, \ldots, x_{i_k}\} \cup \{y_j : e_j \text{ es incidente con un único nodo de } V'\}$

    Se tiene que $w(A') = k'$.

  - Recíprocamente, sea $A' \subseteq A$ tal que $w(A') = k'$.
    Sea $A' = \{x_{i_1}, \ldots, x_{i_m}\} \cup \{y_{j_1}, \ldots, y_{j_q}\}$. Se tiene que
    - $V' = \{x_{i_1}, \ldots, x_{i_m}\}$ es un recubrimiento de vértices de G.
    - $m = k$.

■

El problema PARTICIÓN

- *Dado un conjunto, A y una función peso, w, sobre A, determinar si existe una partición $\{B, C\}$ de A tal que $w(B) = w(C)$.*

Proposición 1: **PARTICIÓN** $\in$ **NP**.

Teorema 2: *El problema* **PARTICIÓN** *es* **NP**–*completo*.

Demostración: Veamos que **SUBSET-SUM** $\leq^p$ **PARTICIÓN**.
Consideremos la aplicación $F : E_{\textbf{SUBSET-SUM}} \to E_{\textbf{PARTICIÓN}}$ definida así:

- Sea $(A, w, k) \in E_{\textbf{SUBSET-SUM}}$. Sean a, b dos elementos que no son de A. Sea $A' = A \cup \{a, b\}$. Sea $w' : A' \to \mathbf{N}$ definida así:

$$w'(x) = \begin{cases} w(x) & \text{si } x \in A \\ k+1 & \text{si } x = a \\ w(A) - k + 1 & \text{si } x = b \end{cases}$$

- Se define $F(A, w, k) = (A', w')$

Para cada $(A, w, k) \in E_{\textbf{SUBSET-SUM}}$ se verifica que

$$(A, w, k) \in L_{\textbf{SUBSET-SUM}} \iff (A', w') \in L_{\textbf{PARTICIÓN}}$$

■

El problema 3-COL

Sea $G = (V, E)$ un un grafo no dirigido y $k \geq 1$.

- Una **coloración** de G **con** k **colores** es una aplicación, f, de V en $\{1, \ldots, k\}$. La imagen $f(u)$ de $u \in V$ es el **color asignado** a u por esa coloración.

- Diremos que un grafo G es k-**coloreable sii** existe una coloración, f, de G con k colores que verifica $f(u) \neq f(w)$, para cada $\{u, w\} \in E$.

Problema 3-COL: **Dado un grafo no dirigido, determinar si es coloreable con 3 colores**.

NP-completitud del problema 3-COL

Proposición 16: 3-COL $\in$ **NP**.

Demostración: Consideremos el siguiente algoritmo $\mathcal{A}$ no determinista:

Entrada: $G = (V, E)$ un grafo no dirigido con $V = \{x_1, \ldots, x_n\}$.

$f \leftarrow \emptyset$

para $i = 1$ hasta n hacer
  Elegir e_1, e_2, e_3
    $e_1 : f \leftarrow f \cup \{(x_i, 1)\}$
    $e_2 : f \leftarrow f \cup \{(x_i, 2)\}$
    $e_3 : f \leftarrow f \cup \{(x_i, 3)\}$

} **Fase no determinista**

para cada $\{u, w\} \in E$ hacer
  si $f(u) = f(w)$ entonces
    devolver no

devolver sí

} **Fase determinista**

Hallemos el coste en tiempo del algoritmo $\mathcal{A}$.

- Tiempo de la fase no determinista: exactamente n pasos (en cada una de las computaciones).
- Tiempo de la fase determinista: cuadrático en el número de nodos del grafo.

El coste en tiempo del algoritmo $\mathcal{A}$ es cuadrático en el número de nodos del grafo de entrada; es decir, lineal en el tamaño de dicho grafo.

Veamos que el algoritmo $\mathcal{A}$ resuelve el problema 3-COL. Para ello, sea G un grafo no dirigido

- Supongamos que G es coloreable con 3 colores. Sea f una coloración válida de G con 3 colores y consideremos la computación $\mathcal{C}$ que en la fase no determinista selecciona, precisamente, la coloración f. Entonces esa computación $\mathcal{C}$ devolverá **sí** y, por tanto, será de aceptación.
- Supongamos que G **no** es coloreable con 3 colores. Sea $\mathcal{C}$ cualquier computación y notemos f' la coloración de G con 3 colores seleccionada en la fase no determinista. Entonces f' **no** será una coloración válida y, por tanto, la computación $\mathcal{C}$ **no** será de aceptación.

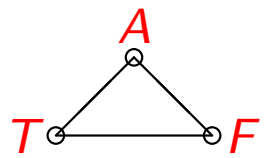
En consecuencia, el problema 3-COL pertenece a la clase **NP**.

Teorema 7: *El problema* 3-COL *es* **NP**–*completo*.

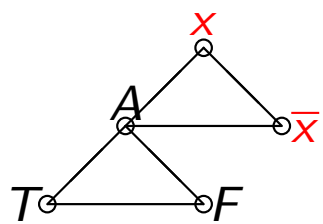
Demostración: Veamos que 3–SAT $\leq^p$ 3-COL.

Para cada $\varphi \in E_{\text{3-SAT}}$ se construye un grafo no dirigido G_φ como sigue:

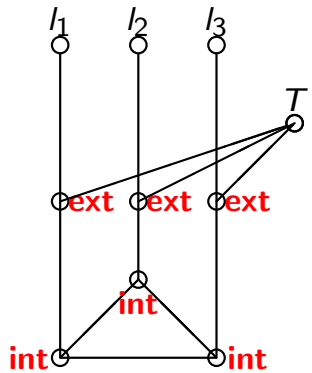
* **Triángulo base** (vértices T, F y A):



* **Cada variable** x de φ proporciona dos nuevos vértices (x y $\overline{x}$):



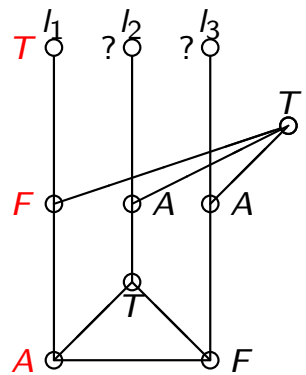
* **Cada cláusula** $c = l_1 + l_2 + l_3$ de φ proporciona seis nuevos vértices.



Vértices añadidos: **externos** e **internos**.

Se considera la aplicación $F : E_{\text{3-SAT}} \to E_{\mathbf{3-COL}}$ definida por $F(\varphi) = G_\varphi$.

* La aplicación F es total computable en tiempo polinomial.
  - Si φ consta de n variables y p cláusulas, entonces el grafo G_φ posee $3 + 2n + 6p$ nodos y $3 + 3n + 12p$ aristas.
* Veamos que φ es satisfacible si y sólo si G_φ es coloreable con 3 colores.
  - Supongamos que φ es satisfactible. Sea σ una valoración tal que $\sigma(\varphi) = 1$. Consideremos en G_φ la siguiente coloración:
    * Triángulo básico: *colores* T, F y A.
    * Vértice asociado a x: color T si $\sigma(x) = 1$ y color F si $\sigma(x) = 0$.
    * Si $c = l_1 + l_2 + l_3$ es cláusula de φ y $\sigma(l_1) = 1$, entonces, el vértice externo adyacente a l_1 lo coloreamos con F, los restantes externos con A y los internos como *correspondan*.

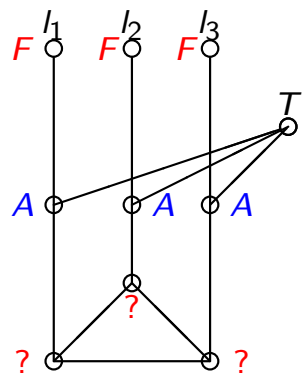


La coloración así definida es un coloreado válido de G.

- Supongamos ahora que G_φ es coloreable con 3 colores (T, F y A).
  - ⋆ En tal situación, las variables de φ sólo pueden estar coloreadas con T o con F (y de forma *compatible*).
  - ⋆ Consideramos la valoración σ definida como sigue:
    - $\sigma(x) = 1$ si x es una variable coloreada con T.
    - $\sigma(x) = 0$ si x es una variable coloreada con F.

    Veamos que esa valoración σ verifica que $\sigma(\varphi) = 1$.

    En efecto: si $\sigma(\varphi) = 0$ entonces existiría una cláusula $c = l_1 + l_2 + l_3$ de φ tal que $\sigma(l_1) = \sigma(l_2) = \sigma(l_3) = 0$. Pues bien, en ese caso no sería posible colorear válidamente los nodos asociados a esa cláusula ya que los nodos externos tendrían que ser coloreados con A y, entonces, sería imposible colorear los tres nodos internos (ya que forman un triángulo).



■

El problema HAM–PATH

Dado un grafo G (dirigido o no dirigido) y dos nodos x, y de G, un **camino hamiltoniano** de x a y en G es un camino **simple** de x a y que pasa por todos los nodos de G.

Un **ciclo hamiltoniano** de G es un ciclo **simple** de G que pasa por todos los nodos de G.

Versión dirigida del camino hamiltoniano con dos nodos distinguidos:

> **Dado un grafo dirigido, G, y dos nodos distinguidos x, y, determinar si existe un camino hamiltoniano de x a y.**

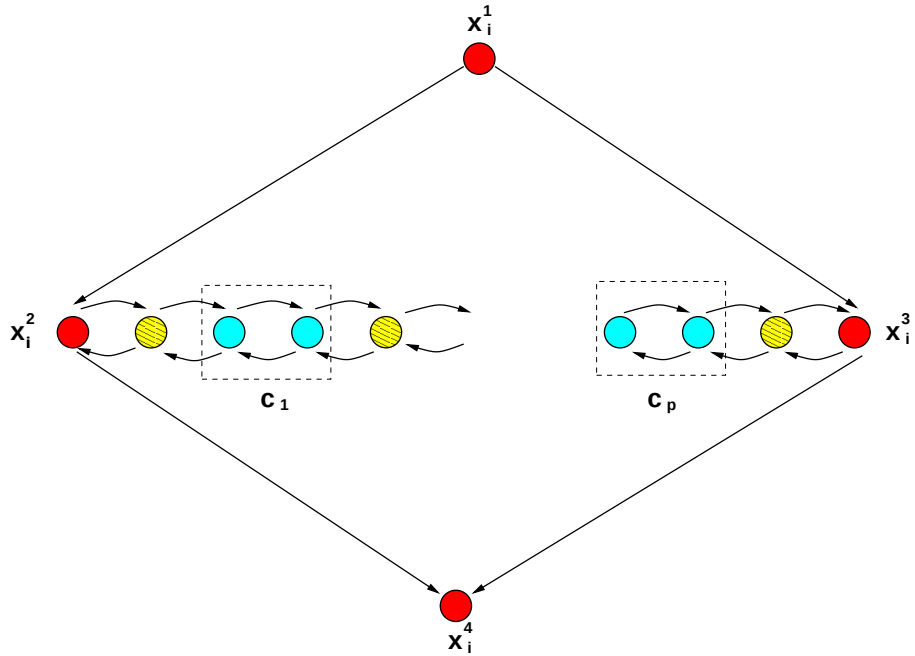
Proposición 1: HAM–PATH $\in$ NP.

Teorema 8: HAM–PATH *es* **NP**–*completo.*

Demostración: Veamos que 3–SAT $\leq^p$ **HAM–PATH**.

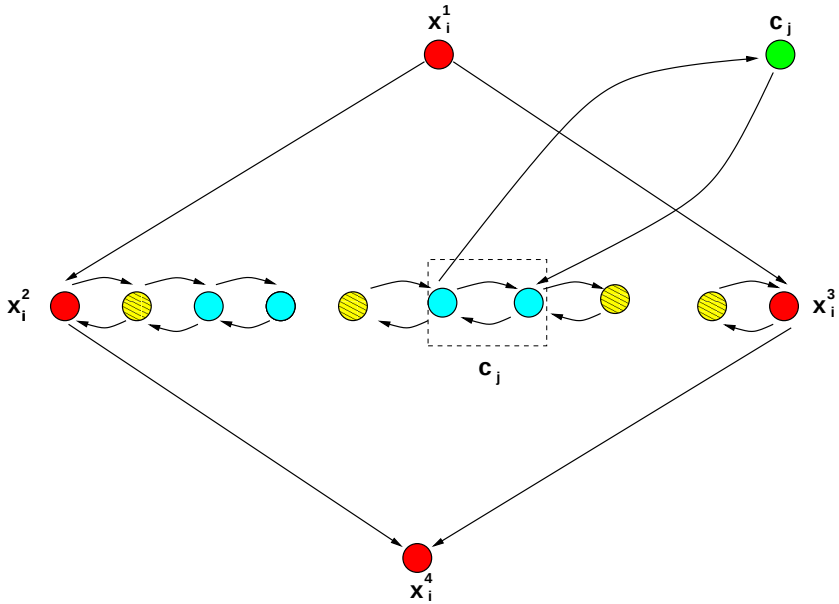
Para cada $\varphi \in E_{3-\mathtt{SAT}}$, con $\mathrm{Var}(\varphi) = \{x_1, \ldots, x_n\}$ se construye un grafo dirigido $G_\varphi = (V_\varphi, E_\varphi)$ así:

- Cada variable, x_i, de φ proporciona un subgrafo de G_φ de estructura de *diamante* así:

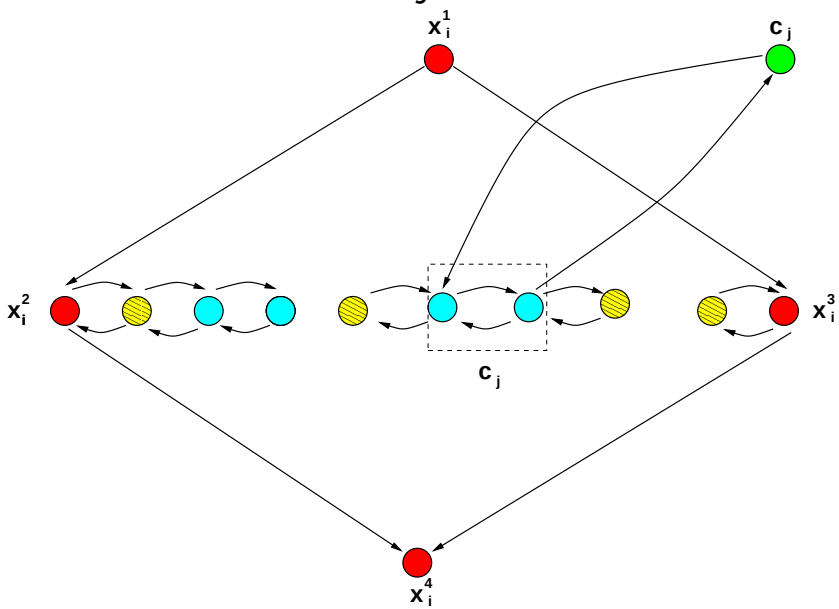


- Para cada i $(1 \leqslant i < n)$, $(x_i^4, x_{i+1}^1) \in E_\varphi$,

- Cada cláusula, c_j, proporciona un nodo conectado en el grafo como sigue:
  - Si x_i aparece en c_j, entonces



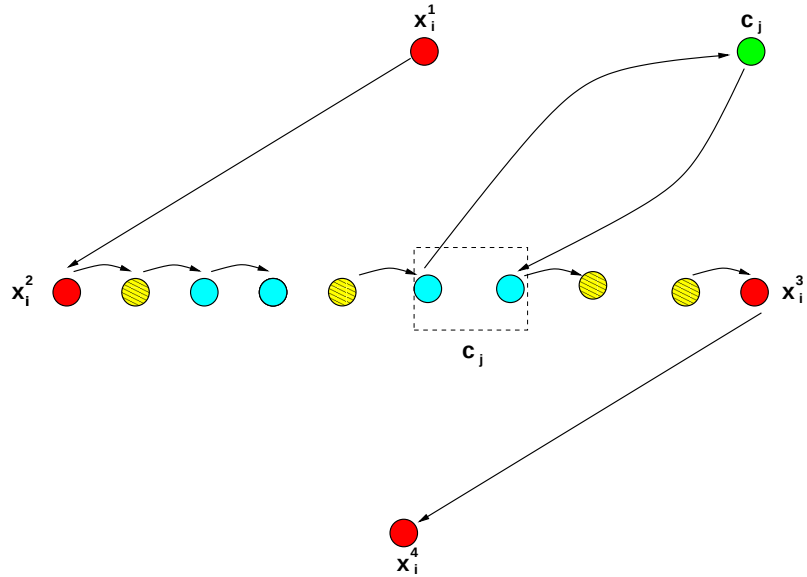
  - Si $\overline{x}_i$ aparece en c_j, entonces



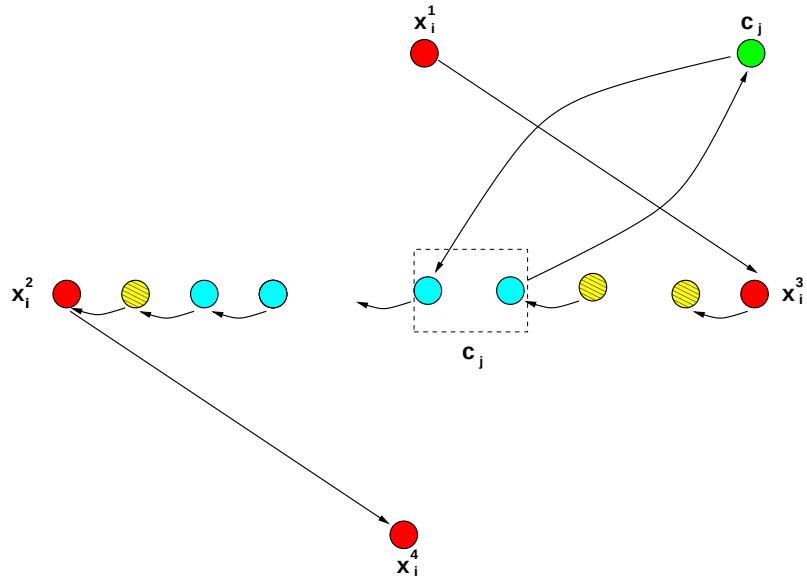
Sea $F : E_{3-\mathtt{SAT}} \to E_{\mathbf{HAM-PATH}}$ definida por $F(\varphi) = (G_\varphi, x_1^1, x_n^4)$.

- F es total computable en tiempo polinomial.
- **Aserto:** *Para cada $\varphi \in E_{3-\mathtt{SAT}}$, φ es satisfactible* **sii** *el grafo G_φ posee un camino hamiltoniano de x_1^1 a x_n^4.*
  - Sean $\varphi = c_1 \ldots c_p$, con $\mathrm{Var}(\varphi) = \{x_1, \ldots, x_n\}$ y σ una valoración tal que $\sigma(\varphi) = 1$. Se construye un camino hamiltoniano de x_1^1 a x_n^4 así:
    - Si $\sigma(x_i) = 1$, entonces se recorre el diamante asociado de izquierda a derecha.
    - Si $\sigma(x_i) = 0$, entonces se recorre el diamante asociado de derecha a izquierda.

- Sea c_j una cláusula y $r_0 = \text{mín}\{r : \ \sigma(l_j^r) = 1\}$.
  Si $l_j^{r_0} = x_i$, entonces se hace el siguiente desvío:



  Si $l_j^{r_0} = \overline{x}_i$, entonces se hace el siguiente desvío:



- Recíprocamente, sea γ un camino hamiltoniano de x_1^1 a x_n^4.
  - ⋆ El camino γ debe ser *normal*: recorre cada diamante bien de izquierda a derecha, o de derecha a izquierda, con la posible salvedad del pequeño atajo para visitar una cláusula.
  - ⋆ Consideramos la valoración σ definida así:

$$\sigma(x_i) = \begin{cases} 1, & \text{si el diamante asociado a } x_i \text{ se recorre de izquierda a derecha} \\ 0, & \text{si el diamante asociado a } x_i \text{ se recorre de derecha a izquierda} \end{cases}$$

Se tiene que $\sigma(\varphi) = 1$.

■

El problema UHAM–PATH

Versión no dirigida del camino hamiltoniano con dos nodos distinguidos:

> **Dado un grafo no dirigido, G, y dos nodos distinguidos x, y, determinar si existe un camino hamiltoniano de x a y.**

Proposición 1: UHAM–PATH $\in$ NP.

Teorema 2: UHAM–PATH *es* **NP**–*completo*.

Demostración: Veamos que **HAM–PATH** $\leq^p$ **UHAM–PATH**.

Sea $(G = (V, E), a, b) \in E_{\textbf{HAM}-\textbf{PATH}}$. Construimos un grafo no dirigido, $G' = (V', E')$, como sigue:

- Cada nodo $u \in V - \{a, b\}$ proporciona tres nodos de G': u^1, u^2, u^3.
- El nodo $a \in V$ proporciona el nodo a^3 de G'.
- El nodo $b \in V$ proporciona el nodo b^1 de G'.
- Para cada $u \in V$, $\{u^2, u^1\} \in E'$ y $\{u^2, u^3\} \in E'$.
- Para cada arco $(u, v) \in E$, se tiene que $\{u^3, v^1\} \in E'$.

Se define $F : E_{\text{HAM}-\text{PATH}} \to E_{\text{UHAM}-\text{PATH}}$ así:

$$F(G, a, b) = (G', a^3, b^1)$$

- F es total computable en tiempo polinomial.
- **Aserto:** *Para cada* $(G, a, b) \in E_{\text{HAM}-\text{PATH}}$, G *posee un camino hamiltoniano de* a *a* b **sii** G' *posee un camino hamiltoniano de* a^3 *a* b^1.
  - Sea $\gamma = (a, u_1, \ldots, u_r, b)$ un camino hamiltoniano de a a b en G.

    Entonces $\gamma' = (a^3, u_1^1, u_1^2, u_1^3, u_2^1, u_2^2, u_2^3, \ldots, u_r^1, u_r^2, u_r^3, b^1)$ es un camino hamiltoniano de a^3 a b^1 en G'.
  - Sea γ' un camino hamiltoniano de a^3 a b^1 en G'.

    Entonces existen nodos $v_1, \ldots, v_r \in V$ tales que

    $\gamma' = (a^3, v_1^1, v_1^2, v_1^3, v_2^1, v_2^2, v_2^3, \ldots, v_r^1, v_r^2, v_r^3, b^1)$.

    Por tanto, $\gamma = (a, v_1, v_2, \ldots, v_r, b)$ es un camino hamiltoniano de a a b en G.

■

El problema UHAM–CYCLE

Versión no dirigida del ciclo hamiltoniano:

Dado un grafo no dirigido, determinar si posee un ciclo hamiltoniano.

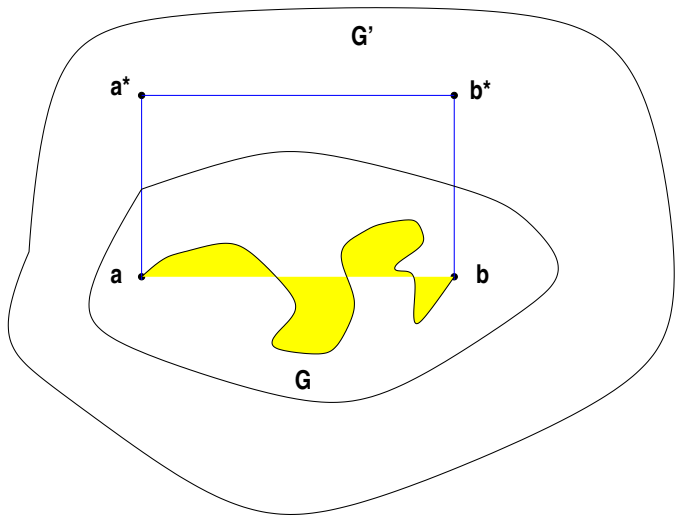
Proposición 1: UHAM–CYCLE $\in$ **NP**.

Teorema 2: UHAM–CYCLE *es* **NP**–*completo*.

Demostración: Veamos que **UHAM–PATH** $\leq^p$ **UHAM–CYCLE**.

Sea $(G = (V, E), a, b) \in E_{\textbf{UHAM–PATH}}$. Construimos un grafo no dirigido, $G' = (V', E')$, como sigue:

- $V' = V \cup \{a^*, b^*\}$, con $a^* \notin V$ y $b^* \notin V$.
- $E' = E \cup \{\{a^*, a\}, \{b^*, b\}, \{a^*, b^*\}\}$.



Se define $F : E_{\text{UHAM-PATH}} \to E_{\text{UHAM-CYCLE}}$ así: $F(G, a, b) = G'$.

- F es total computable en tiempo polinomial.
- **Aserto:** *Para cada* $(G, a, b) \in E_{\text{UHAM-PATH}}$, *$G$ posee un camino hamiltoniano de a a b* **sii** *G' posee un ciclo hamiltoniano.*
  - Sea $\gamma = (a, u_1, \ldots, u_r, b)$ un camino hamiltoniano de a a b en G.

    Entonces $\gamma' = (a^*, a, u_1, \ldots, u_r, b, b^*, a^*)$ es un ciclo hamiltoniano de G'.
  - Sea γ un ciclo hamiltoniano de G'.

    Debido a las conexiones definidas en G', los nodos b, b^*, a^*, a son consecutivos (en ese orden o en el opuesto) en γ.

    Salvo permutaciones, $\gamma = (b, b^*, a^*, a, v_1, \ldots, v_r, b)$.

    Por tanto, $\gamma' = (a, v_1, v_2, \ldots, v_r, b)$ es un camino hamiltoniano de a a b en G.

■

El problema HAM–CYCLE

Versión dirigida del ciclo hamiltoniano:

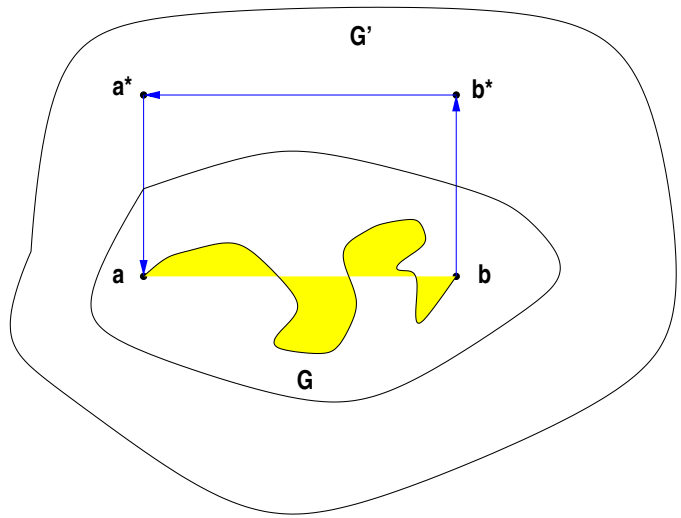
Dado un grafo dirigido, G, determinar si posee un ciclo hamiltoniano.

Proposición 1: HAM–CYCLE $\in$ NP.

Teorema 2: HAM–CYCLE *es* **NP**–*completo*.

Demostración: Veamos que **HAM–PATH** $\leq^p$ **HAM–CYCLE**.

Prueba similar a la anterior. Considerar $E' = E \cup \{(b, b^*), (b^*, a^*), (a^*, a)\}$.



El problema UHAM–PATH*

Versión no dirigida del camino hamiltoniano sin nodos distinguidos:

> **Dado un grafo no dirigido determinar si posee un camino hamiltoniano**

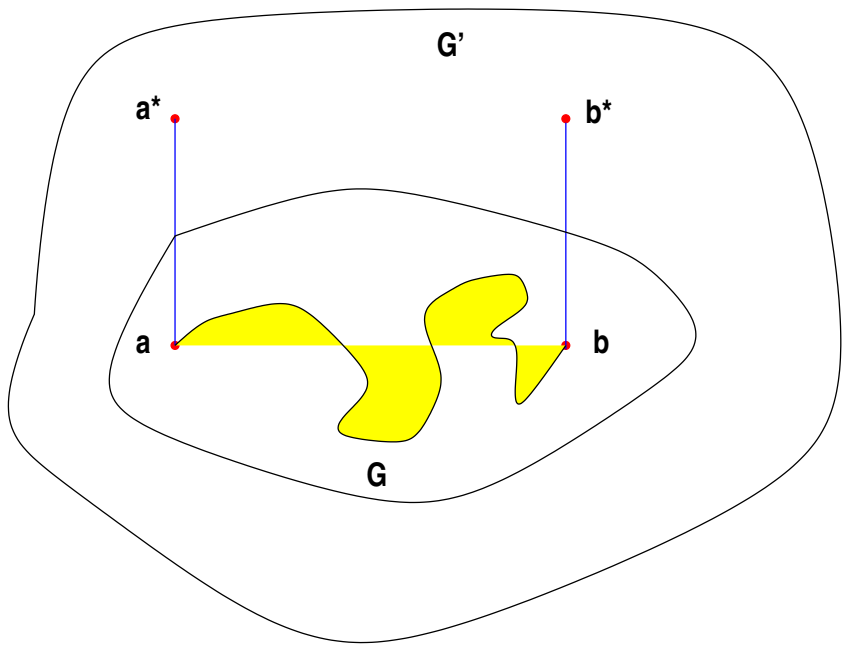
Proposición 1: **UHAM–PATH*** $\in$ **NP**.

Teorema 2: **UHAM–PATH*** *es* **NP***–completo*.

Demostración: Veamos que **UHAM–PATH** $\leq^p$ **UHAM–PATH***.

Sea $(G = (V, E), a, b) \in E_{\text{UHAM}-\text{PATH}}$. Construimos un grafo no dirigido, $G' = (V', E')$, como sigue:

- $V' = V \cup \{a^*, b^*\}$, con $a^* \notin V$ y $b^* \notin V$.
- $E' = E \cup \{\{a^*, a\}, \{b^*, b\}\}$.



Se define $F : E_{\text{UHAM}-\text{PATH}} \to E_{\text{UHAM}-\text{PATH}^*}$ así: $F(G, a, b) = G'$.

- F es total computable en tiempo polinomial.
- **Aserto:** *Para cada* $(G, a, b) \in E_{\text{UHAM}-\text{PATH}}$, G *posee un camino hamiltoniano de* a *a* b **sii** G' *posee un camino hamiltoniano.*
  - Sea $\gamma = (a, u_1, \ldots, u_r, b)$ un camino hamiltoniano de a a b en G. Entonces $\gamma' = (a^*, a, u_1, \ldots, u_r, b, b^*)$ es un camino hamiltoniano de G'.
  - Sea γ un camino hamiltoniano de G'.
    Entonces a^* y b^* deben ser extremos de γ. Luego,
    $\gamma = (a^*, a, v_1, \ldots, v_r, b, b^*)$, o bien γ es el "inverso".
    Por tanto, $\gamma = (a, v_1, v_2, \ldots, v_r, b)$ es un camino hamiltoniano de a a b en G.

■

El problema TSP (del viajante de comercio)

Enunciado informal: *Dadas n ciudades, una función distancia entre ellas y un número natural, k, determinar si existe un recorrido (partiendo de una de las ciudades, visitando las restantes una sóla vez, y volviendo a la de partida) de longitud menor o igual que k.*

Enunciado formal: **Sea $C = \{1, \ldots, n\}$, d una aplicación de $C \times C$ en N tal que $d(i,j) = d(j,i)$ $(\forall i, j)$, y $k \in$ N, determinar si existe una permutación, π, de $\{1, \ldots, n\}$ tal que $\sum_{i=1}^{n} d(\pi(i), \pi(i+1)) \leqslant k$, en donde $\pi(n+1) = \pi(1)$.**

Proposición 1: TSP $\in$ NP.

Teorema 2: *El problema* **TSP** *es* **NP***–completo.*

Demostración: Veamos que **UHAM–CYCLE** $\leq^p$ **TSP**.

Sea $G = (V, E)$ un grafo no dirigido, con $V = \{1, \ldots, n\}$. Sea $C = V$ y $d : C \times C \to \mathbf{N}$ definida así:

$$d(i,j) = \begin{cases} 0 & \text{si } \{i,j\} \in E \\ 1 & \text{e.c.o.c.} \end{cases}$$

Sea $F : E_{\mathbf{UHAM-CYCLE}} \to E_{\mathbf{TSP}}$ definida por $F(G) = (C, d, 0)$.

- F es total computable en tiempo polinomial.
- **Aserto:** *Para cada $G \in E_{\mathbf{UHAM-CYCLE}}$, G posee un ciclo hamiltoniano* **sii** *existe una permutación de $\{1, \ldots, n\}$ tal que $\sum_{i=1}^{n} d(\pi(i), \pi(i+1)) \leqslant 0$ ($n = |G|$).*
  - Sea $G = (V, E)$ un grafo no dirigido, con $V = \{1, \ldots, n\}$.
    - Sea $\gamma = (u_1, \ldots, u_n, u_1)$ un ciclo hamiltoniano de G. Sea π la permutación de $\{1, \ldots, n\}$ definida por $\pi(i) = u_i$ $(1 \leqslant i \leqslant n)$.

      Se tiene que $\sum_{i=1}^{n} d(\pi(i), \pi(i+1)) \leqslant 0$.
    - Sea π una permutación de $\{1, \ldots, n\}$ tal que

      $$\sum_{i=1}^{n} d(\pi(i), \pi(i+1)) \leqslant 0$$

      Entonces $\gamma = (\pi(1), \ldots, \pi(n), \pi(1))$ es un ciclo hamiltoniano de G.

■