Tema 4: NP-completitud

Serafín Moral

Universidad de Granada

Mayo, 2024

Contenido

- Reducción
- Completitud
- El problema de la consistencia en lógica proposicional
- El teorema de Cook
- Variantes de SAT
- Problemas de conjuntos
- Problemas de grafos
- Problemas numéricos
- Otros problemas
- Caracterización de problemas en NP
- Técnicas de reducción de problemas
- Clase CoNP
- Problemas de funciones FNP

Recordatorio: Problemas y Lenguajes

- Recordemos que un problema computacional de decisión $P(x)$ es equivalente a un lenguaje:

$$\{x \in A^* : P(x) =' SI'\}$$

- Y que un lenguaje L sobre A^* define un problema:

Dada $x \in A^*$, ¿Pertenece x a L?

- La teoría se suele explicar en términos de lenguajes y la práctica en términos de problemas, PERTENCEL(x):
- Toda definición en términos de lenguajes tiene una definición equivalente en términos de problemas de decisión y recíprocamente.
- Para acortar la distancia entre teoría y la práctica, en la mayoría de los casos vamos a expresar las definiciones sólo en términos de problemas.

Reducción en términos de problemas

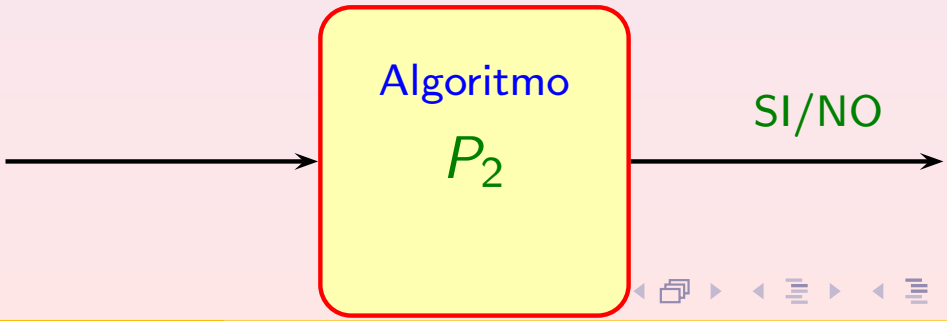
Un problema de decisión $P_1(x)$ es reducible a un problema de decisión $P_2(y)$ donde $x \in X, y \in Y$,

$$P_1(x) \propto P_2(y)$$

si y solo si existe un algoritmo determinista que en espacio logarítmico calcula una función $REDU : X \to Y$ de tal manera que

$$P_1(x) = P_2(REDU(x))$$

Problema $P_1(x)$ reducible a $P_2(y)$: Algoritmo $P_2 \to$ Algoritmo P_1



Reducción en términos de problemas

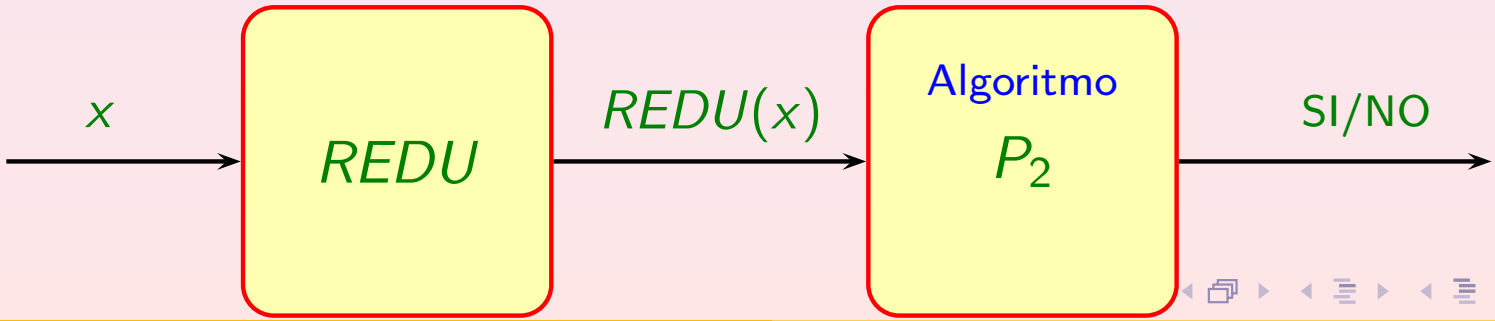
Un problema de decisión $P_1(x)$ es reducible a un problema de decisión $P_2(y)$ donde $x \in X, y \in Y$,

$$P_1(x) \propto P_2(y)$$

si y solo si existe un algoritmo determinista que en espacio logarítmico calcula una función $REDU : X \to Y$ de tal manera que

$$P_1(x) = P_2(REDU(x))$$

Problema $P_1(x)$ reducible a $P_2(y)$: Algoritmo $P_2 \to$ Algoritmo P_1



Reducción en términos de problemas

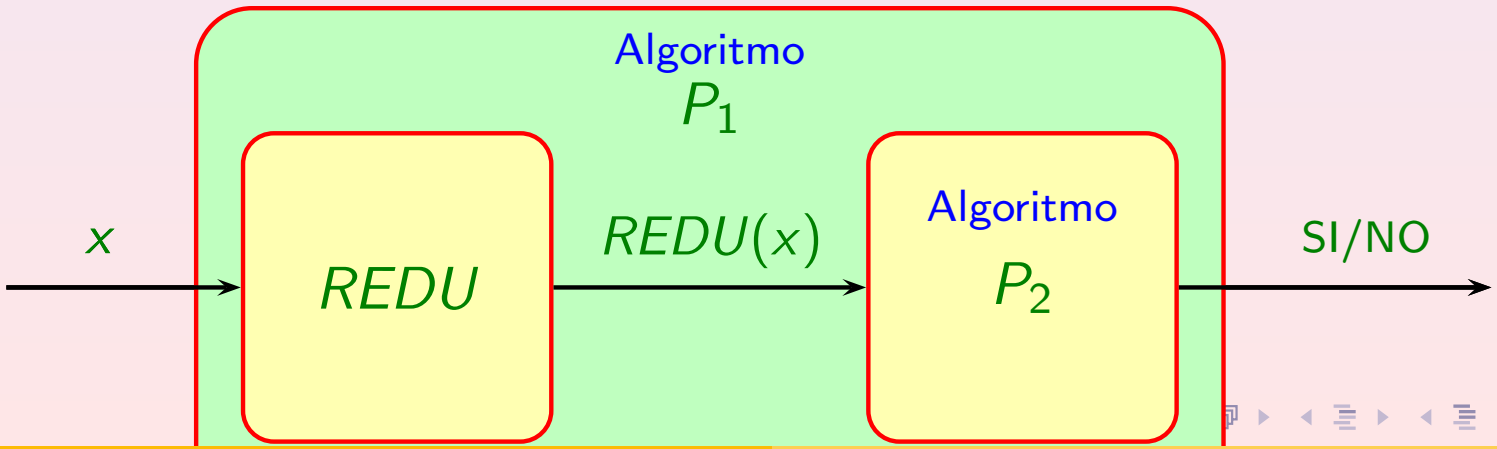
Un problema de decisión $P_1(x)$ es reducible a un problema de decisión $P_2(y)$ donde $x \in X, y \in Y$,

$$P_1(x) \propto P_2(y)$$

si y solo si existe un algoritmo determinista que en espacio logarítmico calcula una función $REDU : X \to Y$ de tal manera que

$$P_1(x) = P_2(REDU(x))$$

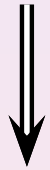
Problema $P_1(x)$ reducible a $P_2(y)$: Algoritmo $P_2 \to$ Algoritmo P_1



Significado de la Reducción

La reducción de P_1 a P_2, indica que si obtuviésemos una solución *sencilla* para P_2, entonces componiéndola con la reducción, podríamos obtener una solución para P_1.

Eso significa que si obtuviésemos una solución *sencilla* para P_2 la tendríamos para P_1, pero lo contrario no tiene por qué verificarse.



En definitiva, P_2 es *al menos tan difícil* como P_1: se resiste más a que se encuentre una solución *sencilla*

Es una forma de comparar problemas.

Reducciones Espacio Logarítmicas

- Nosotros hemos supuesto que las reducciones han de ser espacio logarítmicas.
- En otros textos se supone que son en tiempo polinómico.
- La idea es que la reducción (transformación entre problemas) tiene que poder realizarse de forma "eficiente".
- En realidad las reducciones que aparecen en los textos que suponen que las reducciones se pueden hacer en tiempo polinómico son también espacio logarítmicas.
- Para nosotros la reducción tiene que poder realizarse de forma "muy rápida", que lo concretamos como espacio logarítmico.

Pasos en la Reducción de Problemas

Si tenemos dos problemas $P_1(x)$ y $P_2(y)$ para demostrar que $P_1(x) \propto P_2(y)$ hay que:

1. Dar un algoritmo $REDU$ que transforme cada entrada, x, de P_1 en una entrada, $y = REDU(x)$, del problema P_2.
2. Comprobar que dicho algoritmo es logarítmico en espacio.
3. Comprobar que si $P_1(x)$ tiene solución positiva, entonces $P_2(y)$ también la tiene.
4. Comprobar que si $P_2(y)$ tiene solución positiva, entonces $P_1(x)$ también la tiene.

El Problema del Viajante de Comercio (VC) Versión decisión

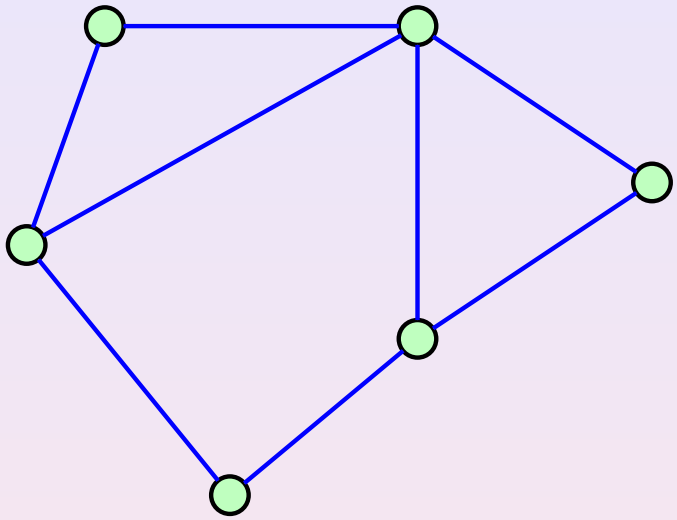
Datos:

- Un conjunto finito de ciudades $C = \{c_1, \ldots, c_m\}$
- Una función de distancia $d : C \times C \to \mathbb{N}$
- Una cota $B \in \mathbb{N}$

Pregunta: ¿Existe un circuito que visite todas las ciudades una sola vez y de coste no superior a B?
Es decir, determinar si existe un orden de las ciudades $(c_{\pi(1)}, \ldots, c_{\pi(n)})$ tal que

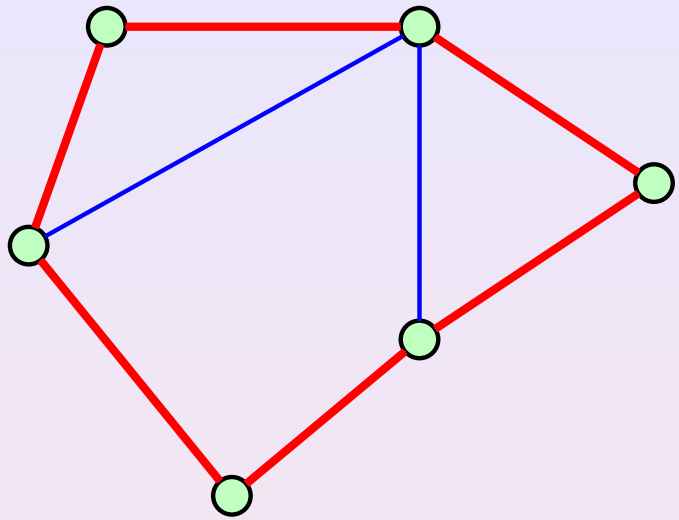
$$\left(\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \right) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B$$

El Problema del Circuito Hamiltoniano (CH)



Datos: Un grafo no dirigido $G = (V, E)$.
Pregunta: ¿Existe un circuito hamiltoniano?

El Problema del Circuito Hamiltoniano (CH)



Datos: Un grafo no dirigido $G = (V, E)$.

Pregunta: ¿Existe un circuito hamiltoniano?

Un circuito hamiltoniano es un camino que parte de un nodo para llegar a él mismo, visitando todos los nodos del grafo una y solo una vez.

Reducción de CH a VC

Reducción del Circuito Hamiltoniano al problema del Viajante de Comercio: $CH \propto VC$

Supongamos una ejemplo del Circuito Hamiltoniano $G = (V, E)$ con $|V| = m$.

Construimos el siguiente ejemplo del Viajante de Comercio:

- Ciudades $C = V$
- Distancia:

$$d(v_i, v_j) = \begin{cases} 1 & \text{si } (v_i, v_j) \in E \\ 2 & \text{si } (v_i, v_j) \notin E \end{cases}$$

- Cota: $B = m$

Comprobaciones

Se puede comprobar que la solución del problema del viajante de comercio es siempre mayor o igual a m y que es m si y solo si desde cada ciudad a la siguiente existe un arco en el grafo original.

- La transformación se puede calcular en espacio logarítmico.
- Si existe un circuito hamiltoniano existe un orden de viaje de valor menor o igual a m (el correspondiente al circuito hamiltoniano).
- Si existe un orden de viaje de valor menor o igual a m, entonces todos los arcos tienen coste 1, y dicho circuito es hamiltoniano en el grafo original.

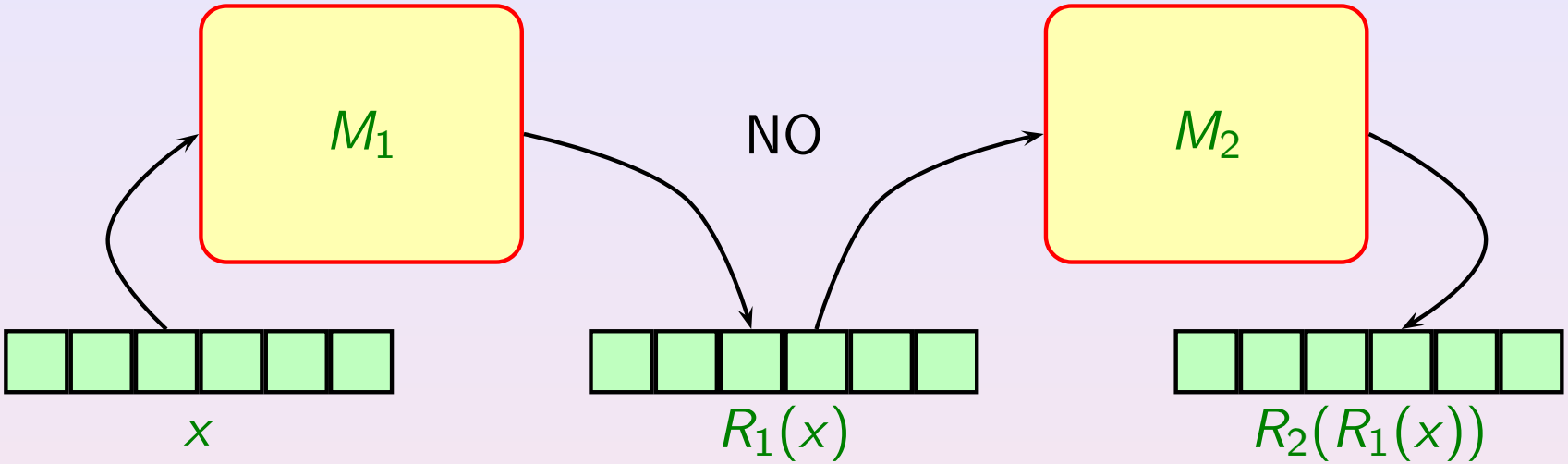
Composición de Reducciones

Si $P_1 \propto P_2$ y $P_2 \propto P_3$, entonces $P_1 \propto P_3$.

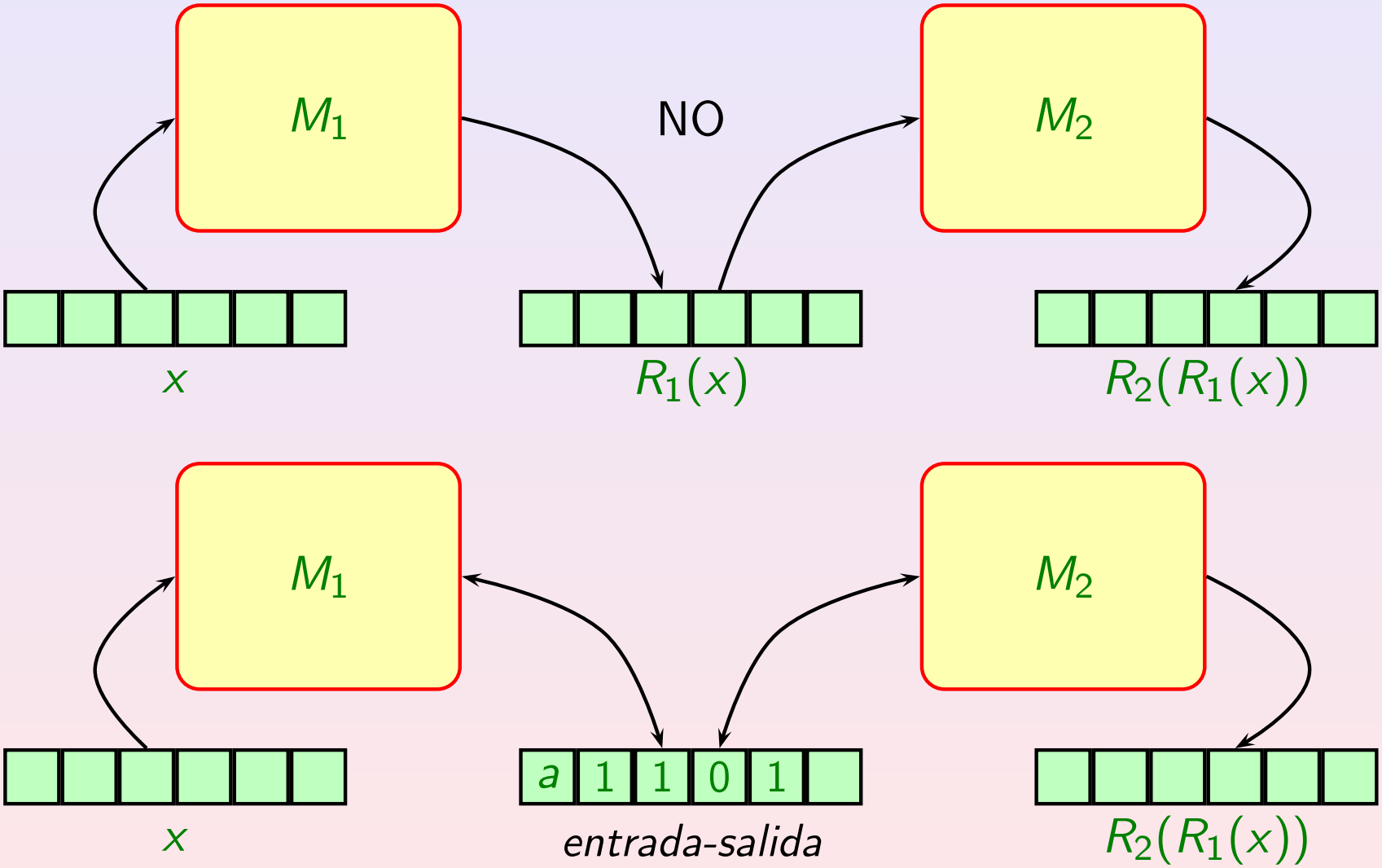
La demostración se basa en la composición de las reducciones de P_1 a P_2 (mediante P_1 calculada por M_1) y de P_2 a L_3 (mediante R_2 calculada por M_2).
La composición no puede ser poner a trabajar M_2 sobre la salida de M_1: $R_1(x)$: Entonces usaríamos un espacio intermedio polinómico para almacenar $M_1(x)$ y la complejidad de la composición en espacio sería polinómica.
La forma de funcionar es que cada vez que M_2 necesita una casilla se la pide a M_1 que la calcula en ese momento, usando un contador binario para determinar la casilla exacta.

P_1 y P_2 se dicen equivalentes si y solo si $P_1 \propto P_2$ y $P_2 \propto P_1$.

Composición en Espacio



Composición en Espacio



Problemas NP-completos

Problemas NP-completo

Un problema es NP-Completo si y solo si es un problema de NP y cualquier otro problema de NP se reduce a él.

Son dos condiciones la primera es fácil de comprobar, la segunda no.

Los problemas NP-completos son los *más difíciles* o *típicos* dentro de la clase NP.

Dada la definición, 'a priori' no tendría porqué existir ningún lenguaje NP-completo.

Esta definición se extiende a otras clases:

Problemas P-completo

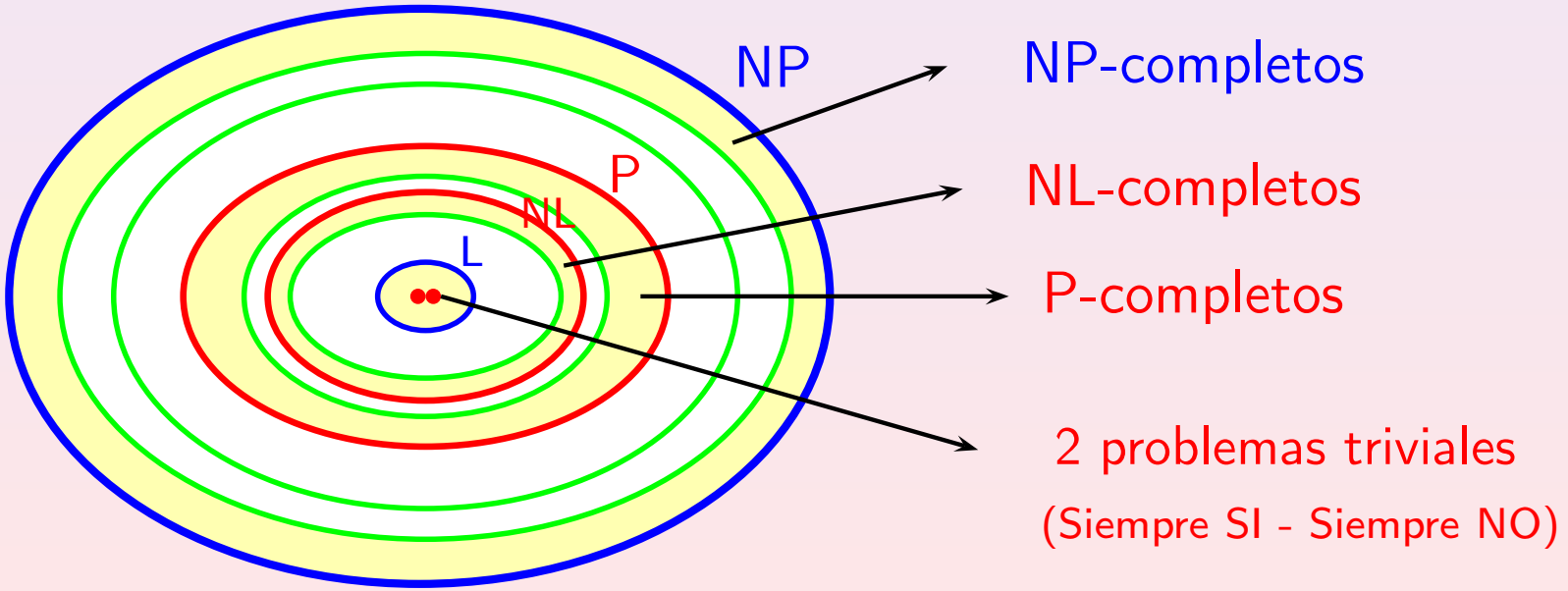
Un problema es P-Completo si y solo si es un problema de P y cualquier otro problema de P se reduce a él.

Equivalencia

Dos problemas P_1 y P_2 son equivalentes si y solo si

$$P_1 \propto P_2 \text{ y } P_2 \propto P_1$$

Esta es una relación de equivalencia que divide el conjunto de los problemas en clases de equivalencia.
En particular esta relación sobre **NP** queda de la siguiente forma:



Consistencia en Lógica Proposicional (SAT)

Datos: Un conjunto $U = \{p_1, \ldots, p_m\}$ de símbolos proposicionales y una colección C de cláusulas sobre estos símbolos.
PREGUNTA: ¿Son consistentes las cláusulas?
EJEMPLO:

- $U = \{p_1, p_2\}$, $C = \{p_1 \vee \neg p_2, \neg p_1 \vee p_2\}$
  La respuestas es SI, ya que todas las cláusulas se satisfacen haciendo p_1 y p_2 verdaderas. En notación matemática $t(p_1) = t(p_2) = V$.

Ejemplo

- $U = \{p_1, p_2\}$, $C = \{p_1 \vee p_2, p_1 \vee \neg p_2, \neg p_1\}$
  En este caso, la respuesta es NO.
  Para satisfacer la tercera cláusula, $t(p_1) = F$ (p_1 es falsa). En estas condiciones, para satisfacer la segunda, tenemos que tener $t(p_2) = F$. Y ahora con p_1 y p_2 falsas, es imposible satisfacer la primera.

Teorema de Cook

El problema de la consistencia en lógica proposicional (SAT) es NP-completo

Demostración:

Lo primero es demostrar que es un problema de NP. Esto es sencillo con el siguiente algoritmo no-determinista:

- Para cada símbolo en U seleccionar un valor de verdad (V ó F).
- Para cada cláusula comprobar si se satisface.
- Si todas se satisfacen responder SI, en caso contrario NO.

Teorema de Cook: Reducción

Ahora hay que demostrar que si un problema, $Pr(x)$, está en NP, entonces dicho lenguaje se reduce a SAT.
Supongamos $Pr(x)$ en NP, la reducción consiste en transformar la entrada al problema x (se suponen una palabra) :
en un ejemplo de SAT tal que
Pr(x)='Si' si y solo si el ejemplo es consistente.
Esta transformación tiene que hacerse en espacio logarítmico.

Reducción

Como $Pr(x)$ está en NP, entonces existe una Máquina de Turing no determinista M que decide $Pr(x)$ en tiempo polinómico (con un polinomio $p(n)$), con un estado de aceptación q_a (para las respuestas positivas) y uno de rechazo q_r (para las respuestas negativas) y una sola cinta (que supondremos ilimitada solo por la derecha).
Supondremos $p(n) \geq n$.
Vamos a construir una transformación que depende de esta máquina:

$$REDU_M : A^* \rightarrow SAT$$

tal que $Pr(x) =' Si' \Leftrightarrow M$ acepta x $\Leftrightarrow REDU_M(x)$ es consistente.

Teorema de Cook: Notación

- Sea $Q = \{q_0, \ldots, q_r\}, \quad r = |Q| - 1, \quad q_1 = q_a, \quad q_2 = q_r$.
- Si $A = \{s_1, \ldots, s_v\}$ hagamos $s_0 = \#$ y $B = \{s_0, s_1, \ldots, s_v\} = \{\#\} \cup A$.
- Sea l el número máximo de opciones de M (supondremos que siempre hay l opciones excepto en los estados de aceptación y rechazo que no hay ninguna).
- Como la máquina da $p(n)$ pasos, el número máximo de casillas visitadas es $p(n) + 1$.

Teorema de Cook: Variables Proposicionales

Hay cuatro tipos de variables:

- $Q[i,k]$, $\quad 0 \leq i \leq p(n), \quad 0 \leq k \leq r$
  En el momento i, la máquina M está en el estado q_k.
- $H[i,j]$, $\quad 0 \leq i \leq p(n), \quad 0 \leq j \leq p(n)$
  En el momento i, la cabeza está en la casilla j.
- $S[i,j,k]$, $\quad 0 \leq i \leq p(n), \quad 0 \leq j \leq p(n), \quad 0 \leq k \leq v$
  En el momento i, la casila j contiene el símbolo s_k
- $O(i,k)$, $\quad 0 \leq i \leq p(n), \quad 1 \leq k \leq l$
  En el momento i, la máquina elige la opción k.

Se supone que si la máquina termina en el instante j, todas las variables permanecen inalteradas en los instantes $i > j$ hasta $p(n)$. El número de variables es polinómico y se pueden calcular en espacio logarítmico.

Grupo 1: Estado de la Máquina

$$Q[i,0] \vee Q[i,1] \vee \cdots \vee Q[i,r], \qquad 0 \leq i \leq p(n)$$

En cada momento, la máquina está en un estado

$$\neg Q[i,j] \vee \neg Q[i,j'], \qquad 0 \leq i \leq p(n), \quad 0 \leq j < j' \leq r$$

En cada momento, la máquina no puede estar a la vez en dos estados distintos

Grupo 2: Cabeza de Lectura

$$H[i,0] \vee H[i,1] \vee \cdots \vee H[i,p(n)], \qquad 0 \leq i \leq p(n)$$

En cada momento, la cabeza de lectura está situada en alguna casilla

---

$$\neg H[i,j] \vee \neg H[i,j'], \qquad 0 \leq i \leq p(n), \quad 0 \leq j < j' \leq p(n)$$

En cada momento, la cabeza de lectura no puede estar en dos casillas distintas

Grupo 3: Casillas

$$S[i,j,0] \vee S[i,j,1] \vee \cdots \vee S[i,j,v], \qquad 0 \leq i \leq p(n), \quad 0 \leq j \leq p(n)$$

En cada momento i, en la casilla j hay algún símbolo del alfabeto.

$$\neg S[i,j,k] \vee \neg S[i,j,k'],$$

$$0 \leq i \leq p(n), \quad 0 \leq j \leq p(n), \quad 0 \leq k < k' \leq v$$

En cada momento i, y en cada casilla j no puede haber dos símbolos distintos.

Grupo 4: Configuración Inicial

- $Q[0,0]$, en el momento inicial (0), la máquina está en el estado 0.
- $H[0,0]$, , en el momento inicial (0), la cabeza está en la casilla 0.
- Si la palabra de entrada es: $x = s_{k_1} \dots s_{k_n}$ entonces se introducen las cláusulas:

$$S[0,0,k_1], S[0,1,k_2], \dots, S[0,n-1,k_n]$$

Inicialmente, los símbolos de la palabra x están en las n primeras casillas de la cinta

Estas son las únicas cláusulas que dependen de la entrada.

- $S[0,n,0], S[0,n+1,0], \dots, S[0,p(n),0]$
Inicialmente, el resto de las casillas de entrada contienen el síbolo $B = s_0$.

Grupo 5: No-Determinismo

$O(i,1) \vee \cdots \vee O(i,l), \quad 0 \leq i \leq p(n)-1.$

En cada momento la máquina toma una opción.

$\neg O(i,j) \vee \neg O(i,j'), \quad 0 \leq i \leq p(n)-1, \quad 1 \leq j,j' \leq l, \quad j \neq j'.$

La máquina no puede tomar dos opciones a la vez en un momento dado.

Si $\delta(q_k, s_d) = \{(q_{k_1}, s_{d_1}, m_1), \ldots, (q_{k_l}, s_{d_l}, m_l)\}$, entonces añadimos todas las cláusulas:

$$\neg H[i,j] \vee \neg Q[i,k] \vee \neg S[i,j,d] \vee \neg O(i,e) \vee Q[i+1,k_e]$$

$$\neg H[i,j] \vee \neg Q[i,k] \vee \neg S[i,j,d] \vee \neg O(i,e) \vee S[i+1,j,d_e]$$

$$\neg H[i,j] \vee \neg Q[i,k] \vee \neg S[i,j,d] \vee \neg O(i,e) \vee H[i+1,j+m_e]$$

donde

$$0 \leq i \leq p(n)-1, \quad 0 \leq j \leq p(n),$$

$$0 \leq k \leq r, \quad 0 \leq d \leq v, \quad 1 \leq e \leq l$$

$$\neg H[i,j] \vee \neg S[i,j',d] \vee S[i+1,j',d]$$

donde

$$0 \leq i \leq p(n)-1, \quad 0 \leq j,j' \leq p(n), (j \neq j')$$

$$0 \leq d \leq v$$

Grupo 8: Funcionamiento de la Máquina, después de parar

Si $\delta(q_k, s_d) = \emptyset$, entonces añadimos todas las cláusulas:

$$\neg H[i,j] \vee \neg Q[i,k] \vee \neg S[i,j,d] \vee Q[i+1,k]$$

$$\neg H[i,j] \vee \neg Q[i,k] \vee \neg S[i,j,d] \vee S[i+1,j,d]$$

$$\neg H[i,j] \vee \neg Q[i,k] \vee \neg S[i,j,d] \vee H[i+1,j]$$

donde

$$0 \leq i \leq p(n)-1, \quad 0 \leq j \leq p(n),$$

$$0 \leq k \leq r, \quad 0 \leq d \leq v, \quad 1 \leq e \leq l$$

Grupo 9: Condición de aceptación

$Q[p(n), 1]$: En el último paso, $p(n)$, la máquina se encuentra en un estado de aceptación.

Equivalencia de los Problemas

Está claro que por la forma de construir la máquina.
Las clásulas reproducen exactamente el funcionamiento de la máquina para la palabra de entrada e incluyen una cláusula que indica que la palabra es aceptada. La libertad está en elegir el valor de verdad de las opciones que puede coger la MT. El resto de las variables tienen un valor de verdad determinado dada la opción elegida. Si al final se puede satisfacer la cláusula $Q[p(n), 1]$ es que se puede llegar a un estado de aceptación.
Entonces la palabra es aceptada si y solo si todas es posible satisfacer todas las cláusulas.

Complejidad de la Reducción

La reducción que hemos realizado tiene una complejidad espacio logarítmica en función de la longitud de x: $|x| = n$.

Primero, n se representa con $t = \log(n)$ bits.
El cálculo de $p(n)$ requiere realizar varias operaciones, multiplicaciones y sumas. El número de operaciones es fijo, por lo que el espacio necesario será del orden de $t = \log(n)$.
$p(n)$ se representará también con un número de cifras que será del orden de $t = \log(n)$.
Finalmente, toda la transformación maneja índices que varían entre 0 y $p(n)$ o entre dos valores constantes. En el segundo caso, el espacio no varía en función de la longitud de x, y en el primer caso se necesita un espacio de orden $t = \log(n)$ (el espacio necesario para el índice más grande) para almacenarlos.
Luego, la complejidad en espacio de toda la transformación es de orden $O(\log(n))$.

Estrategia para NP-Completitud

Supongamos que queremos demostrar que Pr es NP-completo.
Tenemos que hacer:

- Demostrar que Pr está en NP.
- Determinar un problema NP-completo ya conocido Pr', y demostrar que

$$Pr' \propto Pr$$

Como la reducción es transitiva, esta última condición garantiza que cualquier otro problema de NP se reduce a L.

3-SAT

- Hay dos formas de entender 3-SAT: a lo más 3 literates y exactamente 3 literales.
- Las dos son NP-completas.
- Es NP ya que es un caso particular de SAT y éste ya estaba en NP: el mismo algoritmo vale.
- Para demostrar que es completo para la clase NP, no realizamos una demostración similar a la del teorema de Cook. Simplemente, reducimos SAT a 3-SAT.

  Para ello, dado un ejemplo de SAT tenemos que construir un ejemplo de 3-SAT que tenga la misma respuesta: las cláusulas son consistentes o inconsistentes en ambos casos a la vez.

SAT $\propto$ 3-SAT

Supongamos un ejemplo de SAT, con símbolos U y un conjunto de cláusulas C.
Construimos un ejemplo de 3-SAT, que tiene como símbolos los de U y algunos adicionales y cuyas cláusulas, C', se obtienen de C de la siguiente forma:

- Las cláusulas de C de longitud menor o igual a 3 se añaden tal cual a C'.

SAT $\propto$ 3-SAT (Cont.)

La reducción se basa en el siguiente hecho:
Si tengo la fórmula $P \vee Q$ donde P y Q son dos fórmulas, entonces esta formula se puede satisfacer si y solo si se puede satisfacer

$$P \vee x, \qquad Q \vee \neg x$$

donde x es un símbolo proposicional que no aparecía en la fórmula.

SAT $\propto$ 3-SAT (Cont.)

La reducción se basa en el siguiente hecho:
Si tengo la fórmula $P \vee Q$ donde P y Q son dos fórmulas, entonces esta formula se puede satisfacer si y solo si se puede satisfacer

$$P \vee x, \qquad Q \vee \neg x$$

donde x es un símbolo proposicional que no aparecía en la fórmula.
Si estas dos fórmulas son ciertas entonces haciendo resolución en x, deduzco que $P \vee Q$ se puede satisfacer.
Si $P \vee Q$ se puede satisfacer, entonces existe una asignación con la que P o Q es cierta. Entonces si P es cierta, hago x falso y en caso contrario hago x verdadero: está claro que ambas cláusulas se pueden satisfacer.

SAT $\propto$ 3-SAT (Cont.)

- Para cada clásula en C de longitud mayor o igual que 4: $l_1 \vee l_2 \vee \cdots \vee l_k, \quad k \geq 4$, donde $l_1, \ldots, l_k$ son literales, añado símbolos proposicionales $x_2, \ldots, x_{k-2}$ y las cláusulas

$$l_1 \vee l_2 \vee x_2, \quad \neg x_2 \vee l_3 \vee x_3, \quad \cdots \quad , \neg x_{k-3} \vee l_{k-2} \vee x_{k-2}, \quad \neg x_{k-2} \vee l_{k-1} \vee l_k,$$

Equivalencia

La demostración de que esta transformación es una verdadera reducción, se basa en comprobar que para cada cláusula $l_1 \vee l_2 \vee \cdots \vee l_k$, este es el resultado de aplicar la regla básica $k-3$ veces con las variables $x_2, \ldots, x_{k-2}$
La primera se aplica, añadiendo x_2 y transformando en

$$l_1 \vee l_2 \vee x_2, \quad \neg x_2 \vee l_3 \vee l_4 \vee \cdots \vee l_k$$

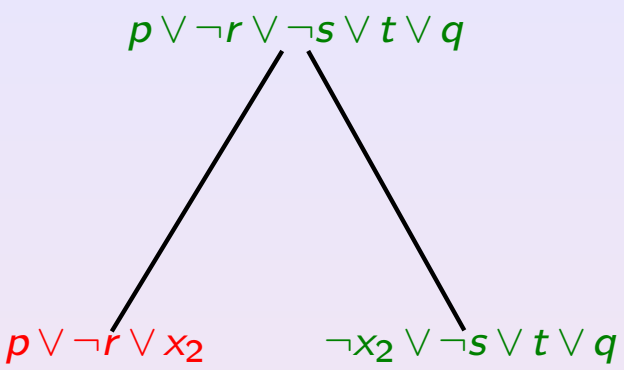
Ahora se aplica recursiva la misma operación a $\neg x_2 \vee l_3 \vee l_4 \vee \cdots \vee l_k$, pero añadiendo cada vez una variable distinta x_k, hasta que nos quedemos con una cláusula de longitud 3.

Ejemplo

$p \vee \neg r \vee \neg s \vee t \vee q$

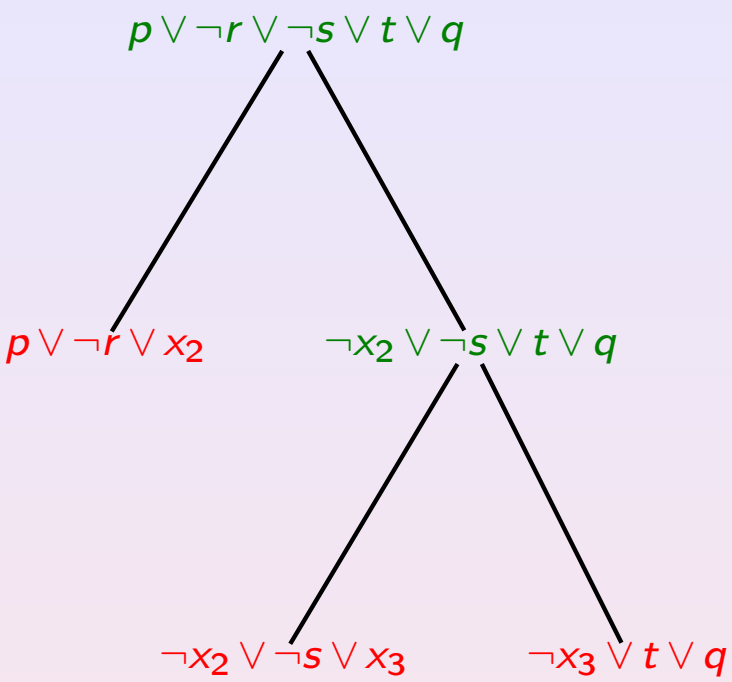
Es importante señalar que para que el algoritmo sea espacio logarítmico, no se puede hacer de forma recursiva calculando las cláusulas intermedias (esas cláusulas pueden ser de tamaño del mismo orden que la entrada). Hay que hacerlo de forma iterativa como se explicó al principio.

Ejemplo



Es importante señalar que para que el algoritmo sea espacio logarítmico, no se puede hacer de forma recursiva calculando las cláusulas intermedias (esas cláusulas pueden ser de tamaño del mismo orden que la entrada). Hay que hacerlo de forma iterativa como se explicó al principio.

Ejemplo



Es importante señalar que para que el algoritmo sea espacio logarítmico, no se puede hacer de forma recursiva calculando las cláusulas intermedias (esas cláusulas pueden ser de tamaño del mismo orden que la entrada). Hay que hacerlo de forma iterativa como se explicó al principio.

3-SAT restringido

Resultado

El problema 3-SAT sigue siendo NP-completo si exigimos que todas las cláusulas tengan exactamente 3 literales distintos.

3-SAT podemos reducirlo a este problema, añadiendo x_1, x_2 y x_3 y añadiendo a toda cláusula de longitud menor que 3, alguna o varias de estas variables.
Por ejemplo $\neg t$ se transformaría en $\neg t \vee x_1 \vee x_2$.
A continuación añadimos cláusulas que fuercen a estas variables a ser falsas: todas las cláusulas de tamaño 3 con estas 3 variables, excepto $x_1 \vee x_2 \vee x_3$. Estas 7 cláusulas solo se pueden satisfacer si x_1, x_2, x_3 son falsas y las nuevas cláusulas son equivalentes a las primeras.

2-SAT esá en NL y, por tanto, en P

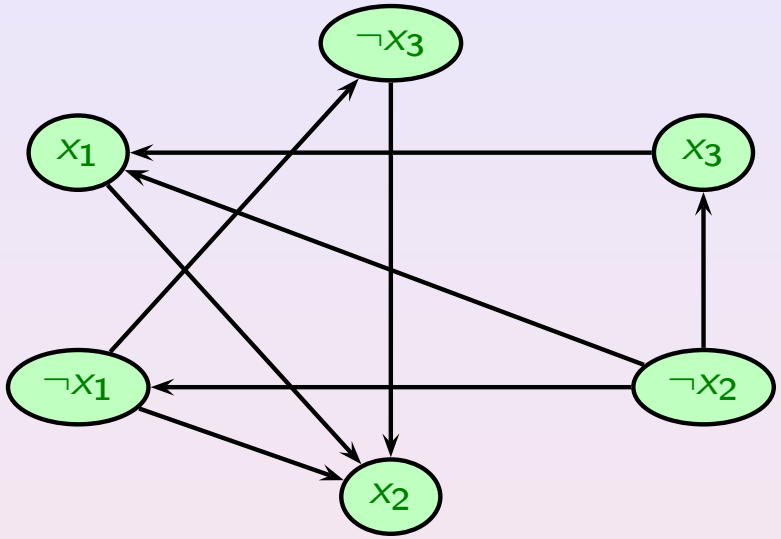
Consideremos un problema 2-SAT, entonces construimos el grafo:

- Nodos: Variables y sus negaciones (todos los posibles literales)
- Arcos: (α, β) es un arco si y solo si $(\neg\alpha \vee \beta)$ está en el problema.

Ejemplo

$$C = \{x_1 \vee x_2, x_1 \vee \neg x_3, \neg x_1 \vee x_2, x_2 \vee x_3\}$$

Grafo:



Teorema

El conjunto de cláusulas C es consistente si y solo si no existe un par de nodos $x, \neg x$, tal que existe un camino de x a $\neg x$ y otro camino de $\neg x$ a x.

Ejemplo

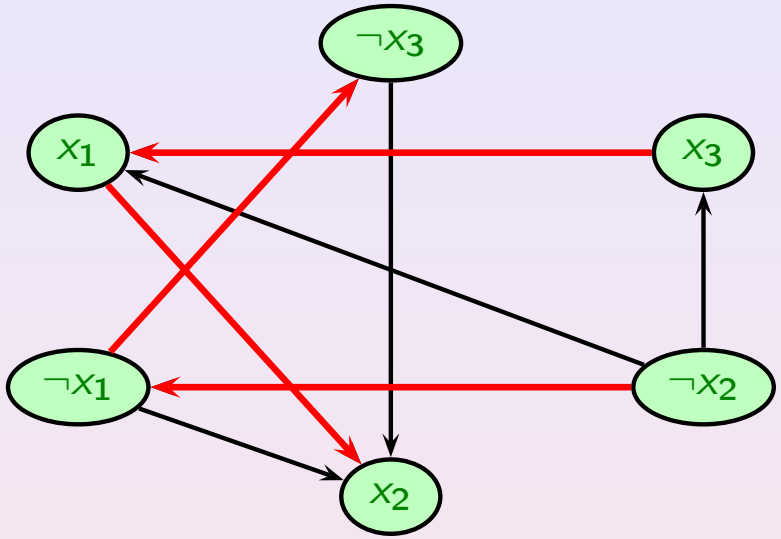
$$C = \{x_1 \vee x_2, x_1 \vee \neg x_3, \neg x_1 \vee x_2, x_2 \vee x_3\}$$

Grafo:

$x_1 \vee \neg x_3$

$\neg x_1 \vee x_2$

$\neg x_3 \vee x_2$



Las deducciones son los caminos.

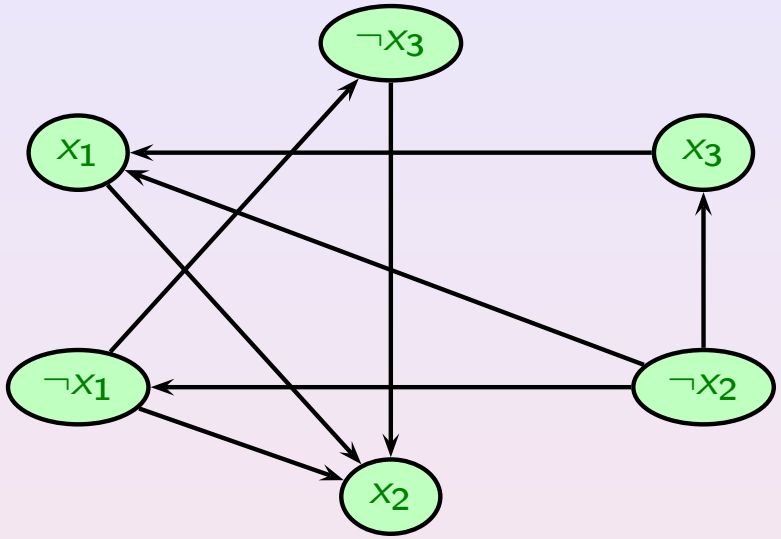
Teorema

El conjunto de cláusulas C es consistente si y solo si no existe un par de nodos $x, \neg x$, tal que existe un camino de x a $\neg x$ y otro camino de $\neg x$ a x.

Ejemplo

$$C = \{x_1 \vee x_2, x_1 \vee \neg x_3, \neg x_1 \vee x_2, x_2 \vee x_3\}$$

Grafo:



Las deducciones son los caminos.

Si hay un camino de x a $\neg x$, se deduce $\neg x$

Si hay un camino de $\neg x$ a x, se deduce x

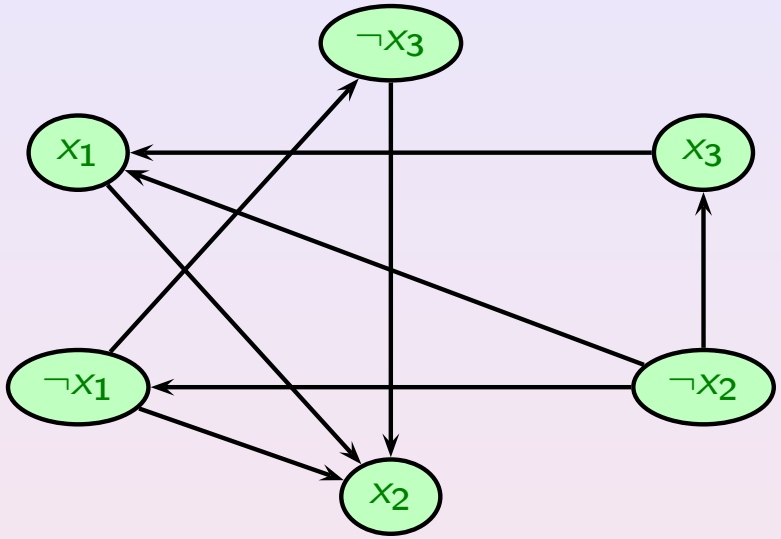
Teorema

El conjunto de cláusulas C es consistente si y solo si no existe un par de nodos $x, \neg x$, tal que existe un camino de x a $\neg x$ y otro camino de $\neg x$ a x.

Ejemplo

$$C = \{x_1 \vee x_2, x_1 \vee \neg x_3, \neg x_1 \vee x_2, x_2 \vee x_3\}$$

Grafo:



Esto demuestra que el complementario de 2-SAT está en NL, pero como NL=CoNL, 2-SAT está en NL

Las deducciones son los caminos.

Si hay un camino de x a $\neg x$, se deduce $\neg x$

Si hay un camino de $\neg x$ a x, se deduce x

Teorema

El conjunto de cláusulas C es consistente si y solo si no existe un par de nodos $x, \neg x$, tal que existe un camino de x a $\neg x$ y otro camino de $\neg x$ a x.

Cláusulas Horn

Son aquellas en las que, a lo más, existe un literal positivo.
Idea del algoritmo polinómico: (V el conjunto de variables proposicionales).

- Se consideran los conjuntos C_1 (todos los literales son negativos) y C_2 (existe un literal positivo).
- Sea H un conjunto de variables proposicionales que inicialmente es igual a las variables que aparecen en las cláusulas de longitud 1 de C_2. La idea es que este conjunto contenga las variables tienen que ser necesariamente verdad.

Algoritmo (Cont.)

- Mientras H cambie, examinar las cláusulas de C_2, si alguna es de la forma $\neg z_1 \vee \neg z_2 \vee \cdots \vee \neg z_i \vee y$ con todas las variables $z_1, \ldots, z_i$ en el conjunto H y la variable y no en H, entonces añadir y a H. (*Como* $\neg z_1 \vee \neg z_2 \vee \cdots \vee \neg z_i \vee y$ *es equivalente a* $(z_1 \wedge z_2 \wedge \cdots \wedge z_i) \rightarrow y$, *si todos los antecedentes son verdaderos, entonces el consecuente también lo tiene que ser*).
- Una vez calculado H se hacen verdad las variables de este conjunto y falsas las de $V \setminus H$.
- La consistencia es equivalente a que para cada cláusula en C_1 exista un literal con su variable en $V \setminus H$.

MAX2SAT

Datos:
Un conjunto de cláusulas con dos literales y un valor $K \geq 0$.
Pregunta:
¿Pueden satisfacerse, al menos, K cláusulas?

Es una generalización de 2-SAT y es NP-completo.
Que es NP es inmediato. Para demostrar que es completo en esta clase, vamos a reducir 3-SAT.

Reducción: 3-SAT $\propto$ MAX2SAT

Para ello cada cláusula de longitud 3, $x \vee y \vee z$ se transforma en las siguientes 10 cláusulas (w es una nueva variable añadida),

$$x, \quad y, \quad z, \quad w, \quad \neg x \vee \neg y, \quad \neg y \vee \neg z, \quad \neg z \vee \neg x,$$
$$x \vee \neg w, \quad y \vee \neg w, \quad z \vee \neg w$$

Se selecciona $K = 7m$, donde m es el número de cláusulas.
Técnica: *Construcción de Gadgets*

Esto se puede comprobar, viendo que para cada cláusula, si $x \vee y \vee z$ es verdadero entonces se puede elegir el valor de verdad de w de manera que se satisfagan 7 cláusulas (pero no más). Si $x \vee y \vee z$ es falso, entonces nunca podemos llegar a satisfacer 7 o más de estas 10 cláusulas, elijamos como elijamos el valor de verdad de w.

Frontera entre P y NP

Muchos problemas, cuando se plantean con suficiente generalidad son NP-completos. Hay restricciones que no llegan a convertirlos en problemas polinómicos, pero siempre, si se restringe el número de casos a resolver lo suficiente se llega un problema polinómico.

Restricción de 3-SAT

3-SAT es NP-completo si hacemos que cada literal nunca aparezca más de dos veces y cada variable más de tres veces, pero pueden existir cláusulas de longitud menor que 3.

Hemos de probar que cualquier ejemplo de 3-SAT se puede transformar en un ejemplo equivalente que cumpla esta restricción. Esto se hace con el siguiente procedimiento:

- Si una variable x no cumple las condiciones de la restricción, apareciendo k veces en las cláusulas, se substituye x por k nuevas variables, $x_1, \ldots, x_k$, una por cada aparición.
- Se añaden las cláusulas: $\neg x_1 \vee x_2, \neg x_2 \vee x_3, \ldots, \neg x_k \vee x_1$
  Estas son equivalentes a: $x_1 \rightarrow x_2, x_2 \rightarrow x_3, \ldots, x_k \rightarrow x_1$
  con lo que todas las versiones x_i tienen que tener el mismo valor de verdad.

Problema NAESAT

Dado un conjunto de clausulas de longitud 3, determinar si existe una asignación de valores de verdad tal que para cada clausula, alguno, pero no todos los literales sean ciertos.

Es claramente NP ya que si se se pueden asignar de forma no determinista valores de verdad a las variables y después comprobar en tiempo polinómico si se verifican las condiciones especificadas: en cada cláusula hay un literal cierto y otro falso.

Reducción de 3-SAT a NAESAT

Se añade una nueva variable z.
Para cada clausula con menos de 3 literales, añadimos z.
Si los tres literales son distintos $p \vee q \vee r$ añadimos una variable s (significado: $p \vee q \leftrightarrow s$) y las cláusulas:

$$s \vee r \vee z, \qquad p \vee q \vee \neg s, \qquad \neg p \vee s \vee z, \qquad \neg q \vee s \vee z$$

Si NAESAT tiene solución, entonces cambiando el valor de verdad de todas las variables, sigue teniendo solución. Elegimos la solución en la que z es falso. Esta es una solución del problema original.

Reciprocamente, si se satisface SAT, podemos satisfacer todas las clausulas nuevas de acuerdo con NAESAT con el mismo valor de verdad para las variables que existían y haciendo z falso y s verdadero si p ó q son verdaderos.

Isomorfismo de Grafos

Problema del Isomorfismo de Grafos

Dados dos grafos no dirigidos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$, determinar si existe un isomorfismo entre G_1 y G_2, esto es una aplicación biyectiva $f : V_1 \to V_2$, tal que $(u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2$.

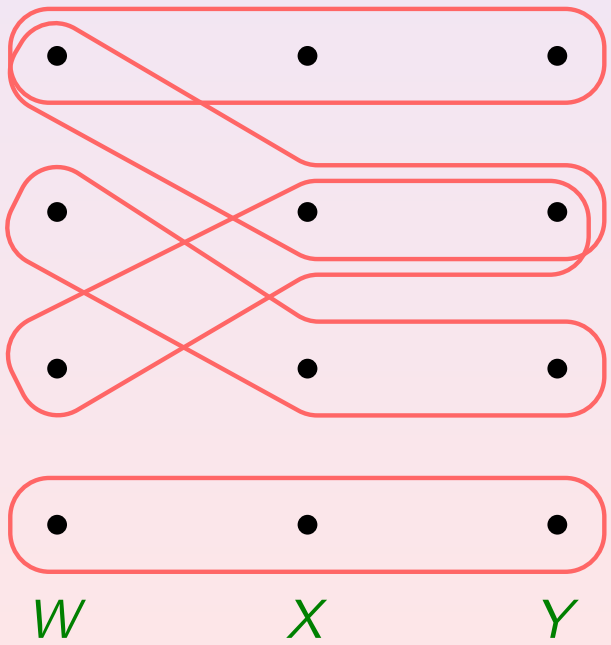
Este es un ejemplo de problema que no se ha demostrado que sea **NP**-completo y tampoco se conoce ningún algoritmo polinómico para resolverlo. Se supone que ni está en **P** ni es **NP**-Completo. De hecho se ha definido una clase **GI** de todos los problemas que se pueden reducir al problema del isoformismo de grafos, de la que evidentemente el problema del isoformismo de grafos es **GI**-completo.

Acoplamiento de Tripletas (ACTRI)

Datos:

- Tres conjuntos disjuntos W, X, Y del mismo tamaño q
- Un subconjunto $M \subseteq W \times X \times Y$ de compatibilidades

Pregunta:
¿Contiene M un subconjunto $M' \subseteq M$ con q elementos, tal que para cada $(w_1, x_1, y_1), (w_2, x_2, y_2) \in M'$, si $(w_1, x_1, y_1) \neq (w_2, x_2, y_2)$, entonces $w_1 \neq w_2, \quad x_1 \neq x_2, \quad y_1 \neq y_2$?.

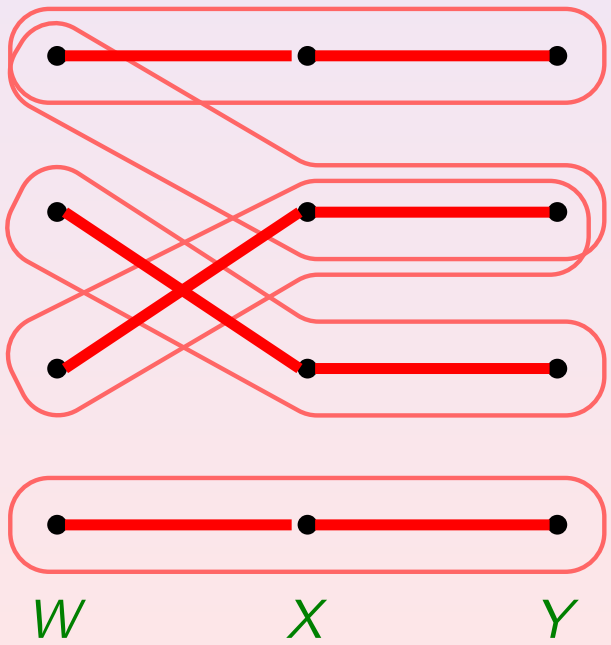


Acoplamiento de Tripletas (ACTRI)

Datos:

- Tres conjuntos disjuntos W, X, Y del mismo tamaño q
- Un subconjunto $M \subseteq W \times X \times Y$ de compatibilidades

Pregunta:
¿Contiene M un subconjunto $M' \subseteq M$ con q elementos, tal que para cada $(w_1, x_1, y_1), (w_2, x_2, y_2) \in M'$, si $(w_1, x_1, y_1) \neq (w_2, x_2, y_2)$, entonces $w_1 \neq w_2, \quad x_1 \neq x_2, \quad y_1 \neq y_2$?.



ACTRI es NP-completo

Es NP, ya que un algoritmo no determinista que elige un subconjunto de M y comprueba que si hay en cada elemento del subconjunto uno y sólo uno de los elementos de cada conjunto tiene tiempo polinómico.

Reducción de 3-SAT a ACTRI

Para demostrar que es completo para NP, para a reducir 3-SAT a ACTRI.
Vamos a suponer un ejemplo de 3-SAT con los siguientes datos:
$U = \{u_1, u_2, \ldots, u_n\}, \qquad C = \{c_1, c_2, \ldots, c_m\}$
Vamos a construir un ejemplo de ACTRI con la misma solución.
Construiremos los conjuntos W, X, Y y el subconjunto de tripletas $M \subseteq W \times X \times Y$.
Para cada variable u_i vamos a introducir en W los elementos $u_i[j], \overline{u}_i[j], \quad j = 1, \ldots, m$, en X los elementos $a_i[j], j = 1, \ldots, m$ y en Y los elementos $b_i[j], j = 1, \ldots, m$.
En M se introducen las tripletas: $T_i^t = \{(\overline{u}_i[j], a_i[j], b_i[j]) : 1 \leq j \leq m\}$ y $T_i^f = \{(u_i[j], a_i[j+1], b_i[j]) : 1 \leq j < m\} \cup \{(u_i[m], a_i[1], b_i[m])\}$.

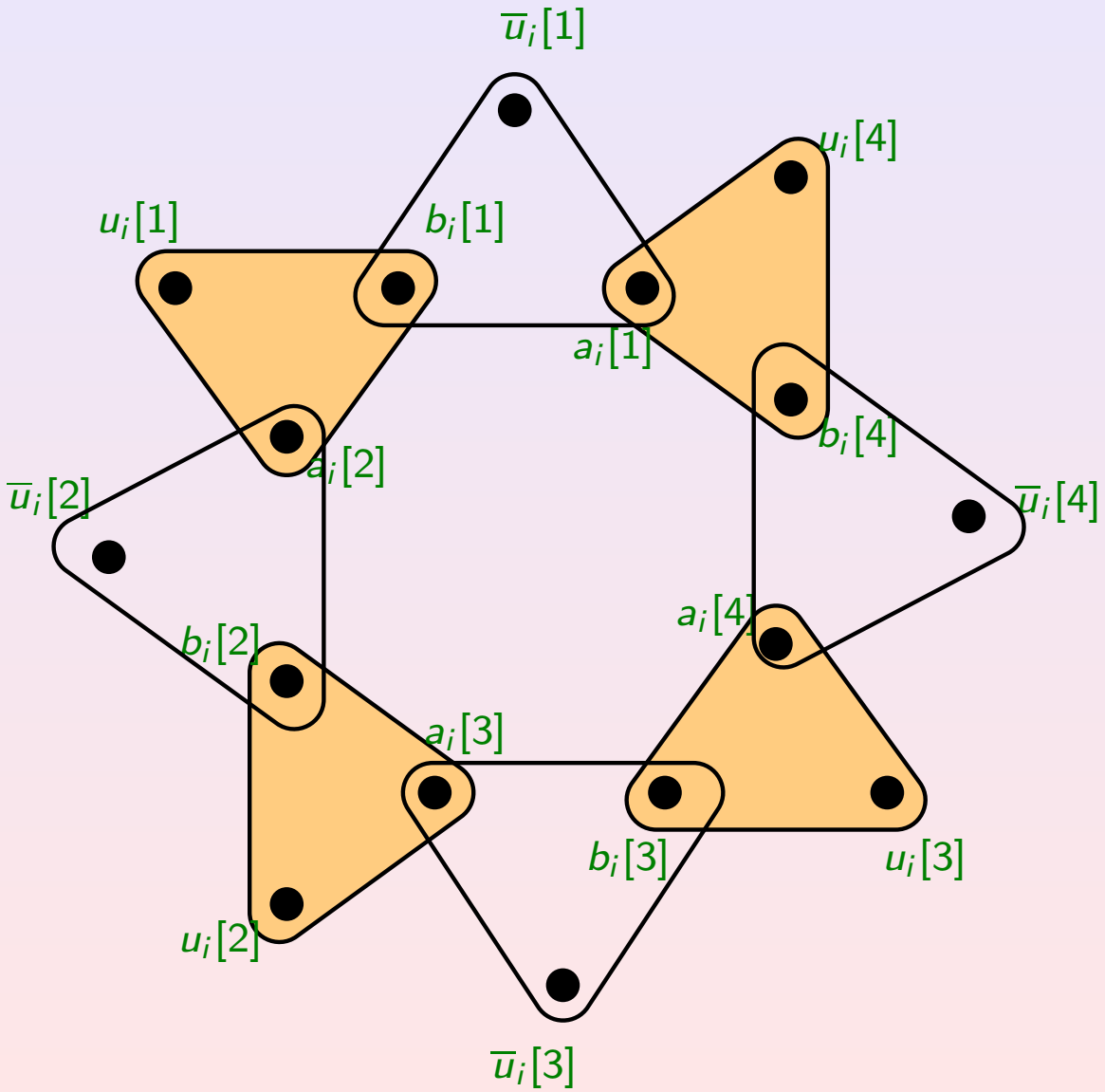
Ejemplo

Ejemplo:
$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}$

$\{(\overline{u}_1[1], a_1[1], b_1[1]), (\overline{u}_1[2], a_1[2], b_1[2]), (u_1[1], a_1[2], b_1[1]), (u_1[2], a_1[1], b_1[2]),$
$(\overline{u}_2[1], a_2[1], b_2[1]), (\overline{u}_2[2], a_2[2], b_2[2]), (u_2[1], a_2[2], b_2[1]), (u_2[2], a_2[1], b_2[2]),$
$(\overline{u}_3[1], a_3[1], b_3[1]), (\overline{u}_3[2], a_3[2], b_3[2]), (u_3[1], a_3[2], b_3[1]), (u_3[2], a_3[1], b_3[2]),$
$(\overline{u}_4[1], a_4[1], b_4[1]), (\overline{u}_4[2], a_4[2], b_4[2]), (u_4[1], a_4[2], b_4[1]), (u_4[2], a_4[1], b_4[2]),$
$\}$

ACTRI es NP-Completo

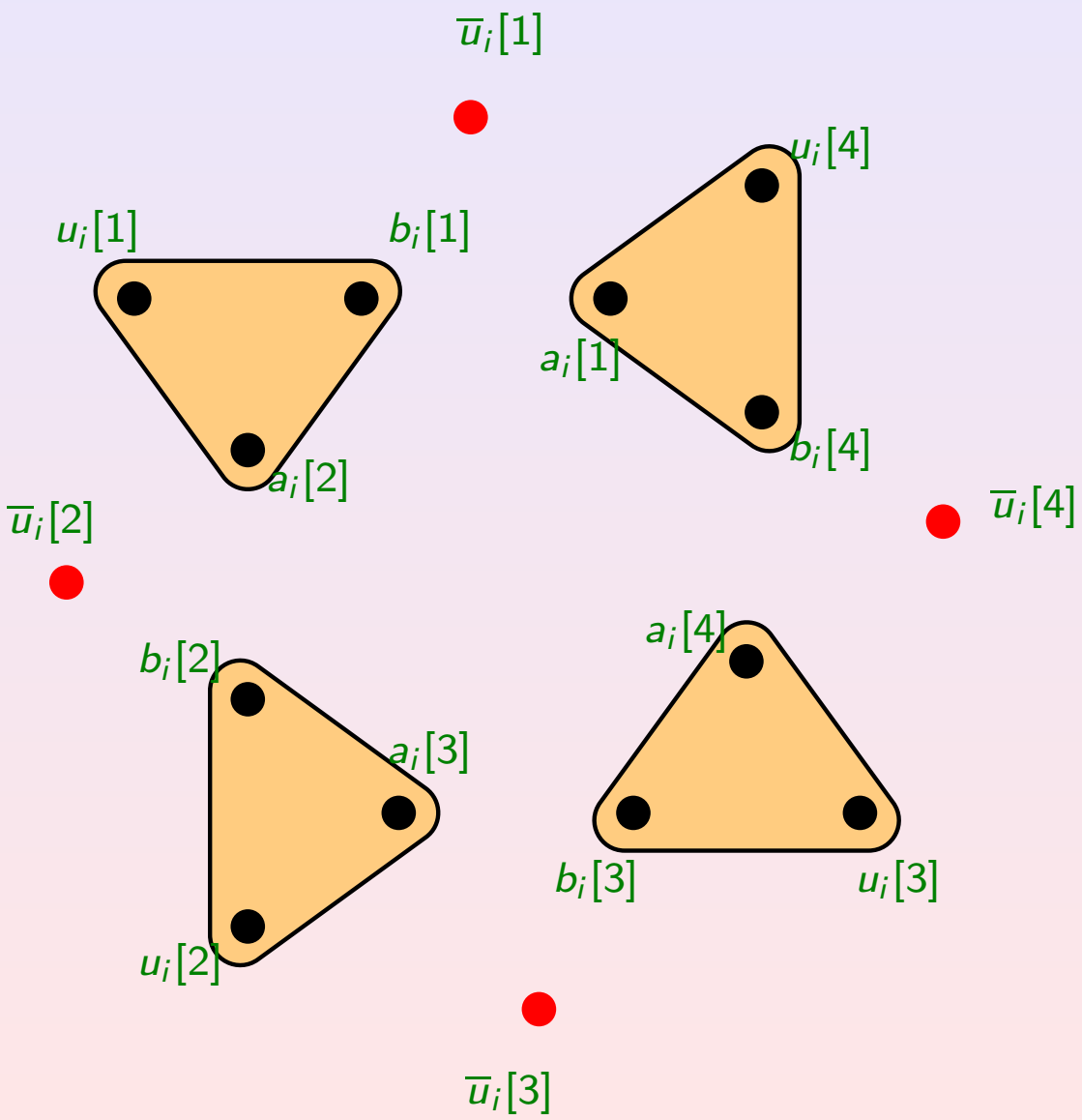
Gráfica de tripletas:



Los valores a_i, b_i no aparecen en otras tripletas. El efecto es que en M' deberán de estas todos los elementos $u_i[1], u_i[2], u_i[3], u_i[4]$ o todos los elementos $\overline{u}_i[1], \overline{u}_i[2], \overline{u}_i[3], \overline{u}_i[4]$.

ACTRI es NP-Completo

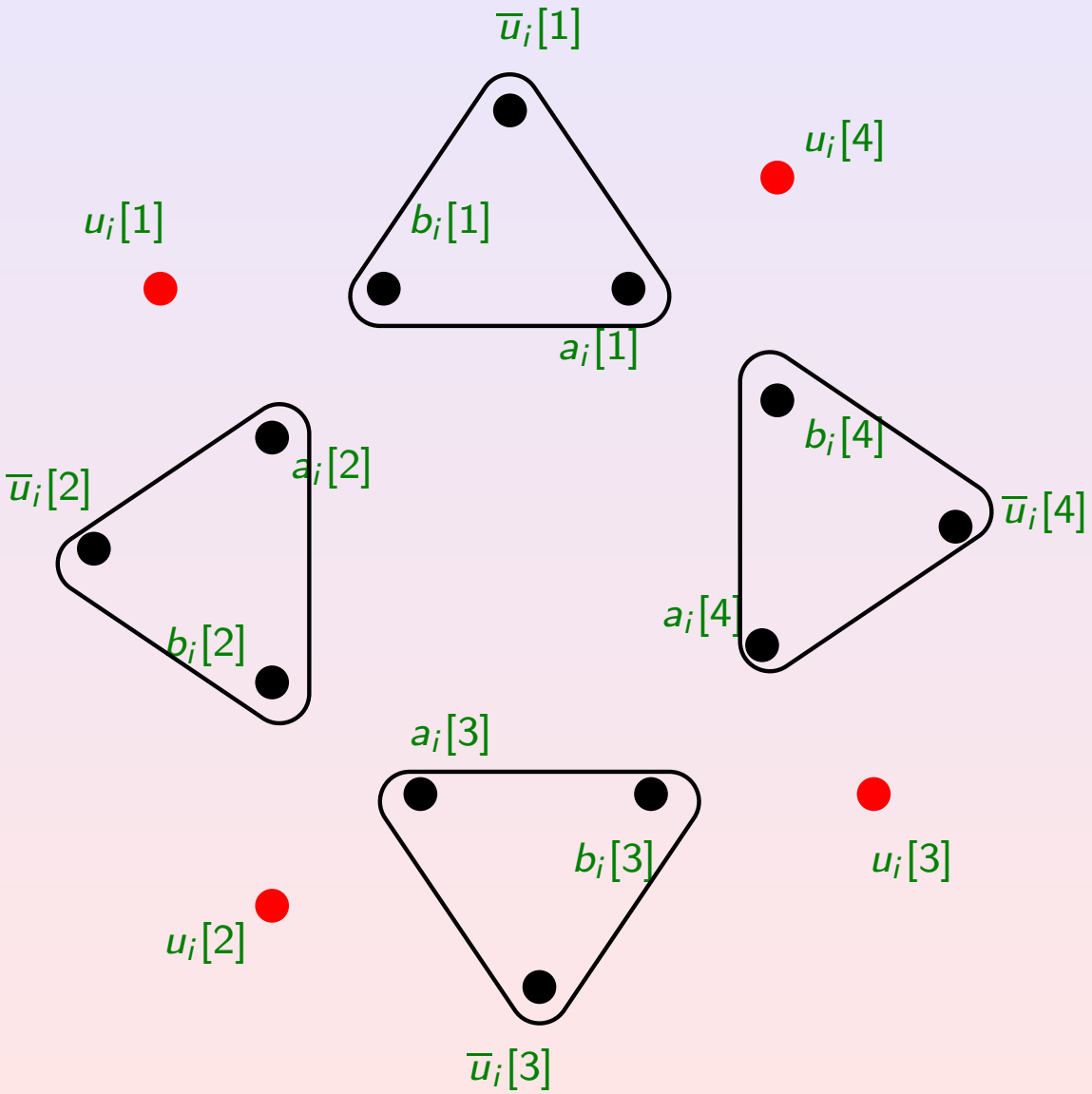
Gráfica de tripletas:



Los valores a_i, b_i no aparecen en otras tripletas. El efecto es que en M' deberán de estas todos los elementos $u_i[1], u_i[2], u_i[3], u_i[4]$ o todos los elementos $\overline{u}_i[1], \overline{u}_i[2], \overline{u}_i[3], \overline{u}_i[4]$.

ACTRI es NP-Completo

Gráfica de tripletas:



Los valores a_i, b_i no aparecen en otras tripletas. El efecto es que en M' deberán de estas todos los elementos $u_i[1], u_i[2], u_i[3], u_i[4]$ o todos los elementos $\overline{u}_i[1], \overline{u}_i[2], \overline{u}_i[3], \overline{u}_i[4]$.

Tripletas para las cláusulas

Por cada cláusula $c_j \in C$:

- Añadir un elemento $s[j]$ a X
- Añadir un elemento $t[j]$ a Y
- Añadir a M las tripletas
  $\{(u_i[j], s[j], t[j]) : u_i \text{ está en } c_j\} \cup$
  $\{(\overline{u}_i[j], s[j], t[j]) : \neg u_i \text{ está en } c_j\}$
  La idea es que en cada acoplamiento M' debe de contener alguna de estas tripletas.
  Si, de las tripletas anteriores, quedaban libres (no elegidos) los elementos $u_i[1], u_i[2], u_i[3], u_i[4]$ corresponde al caso en el que u_i es cierto. Si quedaban libres $\overline{u}_i[1], \overline{u}_i[2], \overline{u}_i[3], \overline{u}_i[4]$, corresponde al caso en el que u_i es falso.
  En consecuencia, estas tripletas garantizan que todas las clásulas se satisfacen.

Elementos adicionales

Se añden los siguientes elementos:

- $g[k], \quad 1 \leq k \leq m(n-1)$ al conjunto X
- $h[k], \quad 1 \leq k \leq m(n-1)$ al conjunto Y
- Al conjunto M las tripletas:

  $(u_i[j], g[k], h[k]), (\overline{u}_i[j], g[k], h[k]),$
  $1 \leq k \leq m(n-1), 1 \leq i \leq n, 1 \leq j \leq m$

  La idea es la siguiente:

  Nos han quedado libres en W después de elegir entre las primeras tripletas: $m.n$ elementos. Después de la segundas tripletas, como hemos elegido uno por cláusula, nos quedan $m.n - m = m(n-1)$. Ahora añadimos este número de elementos a X y a Y y le permitimos que sean compatibles con todos los de X, para tener libertad para elegir los elementos sobrantes en tripletas como queramos.

  El número de tripletas de M es: $2mn + 3m + 2m^2 n(n-1)$

Ejemplo

$$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}$$

$$\begin{aligned} W = \quad & \{u_1[1], u_1[2], \overline{u}_1[1], \overline{u}_1[2], u_2[1], u_2[2], \overline{u}_2[1], \overline{u}_2[2], \\ & u_3[1], u_3[2], \overline{u}_3[1], \overline{u}_3[2], u_4[1], u_4[2], \overline{u}_4[1], \overline{u}_4[2]\} \\ X = \quad & \{a_1[1], a_1[2], a_2[1], a_2[2], a_3[1], a_3[2], a_4[1], a_4[2], s[1], s[2], \\ & g[1], g[2], g[3], g[4], g[5], g[6]\} \\ Y = \quad & \{b_1[1], b_1[2], b_2[1], b_2[2], b_3[1], b_3[2], b_4[1], b_4[2], t[1], t[2], \\ & h[1], h[2], h[3], h[4], h[5], h[6]\} \end{aligned}$$

Ejemplo

$$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}$$

$M =$

$\{(\overline{u}_1[1], a_1[1], b_1[1]), (\overline{u}_1[2], a_1[2], b_1[2]), (u_1[1], a_1[2], b_1[1]), (u_1[2], a_1[1], b_1[2]),$
$(\overline{u}_2[1], a_2[1], b_2[1]), (\overline{u}_2[2], a_2[2], b_2[2]), (u_2[1], a_2[2], b_2[1]), (u_2[2], a_2[1], b_2[2]),$
$(\overline{u}_3[1], a_3[1], b_3[1]), (\overline{u}_3[2], a_3[2], b_3[2]), (u_3[1], a_3[2], b_3[1]), (u_3[2], a_3[1], b_3[2]),$
$(\overline{u}_4[1], a_4[1], b_4[1]), (\overline{u}_4[2], a_4[2], b_4[2]), (u_4[1], a_4[2], b_4[1]), (u_4[2], a_4[1], b_4[2]),$
$(u_1[1], s[1], t[1]), (\overline{u}_3[1], s[1], t[1]), (\overline{u}_4[1], s[1], t[1]),$
$(\overline{u}_1[2], s[2], t[2]), (u_2[2], s[2], t[2]), (\overline{u}_4[2], s[2], t[2]),$
$(u_i[j], g[k], h[k]), (\overline{u}_i[j], g[k], h[k])(i = 1, \ldots, 4, j = 1, 2, k = 1, \ldots, 6)\}$

Solución 3-SAT implica solución ACTRI

Si tenemos una asignación de valores de verdad $t(u_i) \in \{V, F\}$ que satisface todas las cláusulas, elegimos el subconjunto de tripletas M' de la siguiente forma:

- Si u_i es cierto elegimos las tripletas:
  $\{(\overline{u}_i[j], a_i[j], b_i[j]) : 1 \leq j \leq m\}$
- Si u_i es falso:
  $\{(u_i[j], a_i[j+1], b_i[j]) : 1 \leq j < m\} \cup \{(u_i[m], a_i[1], b_i[m])\}$.
- Para cada cláusula j elegimos de entre las tripletas:
  $\{(u_i[j], s[j], t[j]) : u_i \text{ está en } c_j\} \cup$
  $\{(\overline{u}_i[j], s[j], t[j]) : \neg u_i \text{ está en } c_j\}$ elegimos aquella que corresponde a la variable u_i que hacer cierta la cláusula.
- Nos quedan por elegir exactamente $n(m-1)$ elementos de cada conjunto, que se eligen de las tripletas $(u_i[j], g[k], h[k]), (\overline{u}_i[j], g[k], h[k])$. Aquí no hay problema, ya que todos los elementos son compatibles entre sí.

Solución ACTRI implica solución 3-SAT

Supongamos el problema de ACTRI construido a partir de 3-SAT y supongamos que tenemos un subconjunto M' de tripletas solución.

1. Vimos que para cada variable u_i, sólo hay dos formas de elegir las tripletas en una solución: o se eligen las $\{(\overline{u}_i[j], a_i[j], b_i[j]) : 1 \leq j \leq m\}$ o se eligen las $\{(u_i[j], a_i[j+1], b_i[j]) : 1 \leq j < m\} \cup \{(u_i[m], a_i[1], b_i[m])\}$.
2. En el primer caso, hacemos u_i verdadero y en el segundo u_i falso.
3. Para cada cláusula alguna de las 3 tripletas: $\{(u_i[j], s[j], t[j]) : u_i \text{ está en } c_j\} \cup \{(\overline{u}_i[j], s[j], t[j]) : \neg u_i \text{ está en } c_j\}$ debe de haberse elegido en M'.
4. Si hemos elegido $(u_i[j], s[j], t[j])$ con u_i en c_j, entonces, $u_i[j]$ no estaba en las tripletas elegidas consideradas en el paso 1. Por lo tanto, hemos hecho u_i verdadero y la cláusula se satisface.
5. Si hemos elegido $(\overline{u}_i[j], s[j], t[j])$ con $\neg u_i$ en c_j, entonces, $\overline{u}_i[j]$ no estaba en las tripletas elegidas consideradas en el paso 1. Por lo tanto, hemos hecho u_i falso y la cláusula se satisface.

Cubr. por conj. de tamaño tres (3-SET)

Datos: Un conjunto finito X con $|X| = 3q$ y un subconjunto C de subconjuntos de X de tres elementos.
Pregunta: ¿Existe $C' \subseteq C$ tal que $X = \bigcup_{A \in C'} A$ y los elementos de C' son disjuntos dos a dos ($A, B \in C'$ y $A \neq B$, entonces $A \cap B = \emptyset$)?

Es un problema de NP.
Para demostrar que es completo para NP, existe una trasformación muy sencilla: ACTRI $\propto$ 3-SET.
Supongamos un ejemplo de ACTRI dado por W, Y, Z y el subconjunto de tripletas M, construimos el ejemplo de 3-SET donde $X = W \cup Y \cup Z$ y

$$C = \{\{w, y, z\} : (w, y, z) \in M\}$$

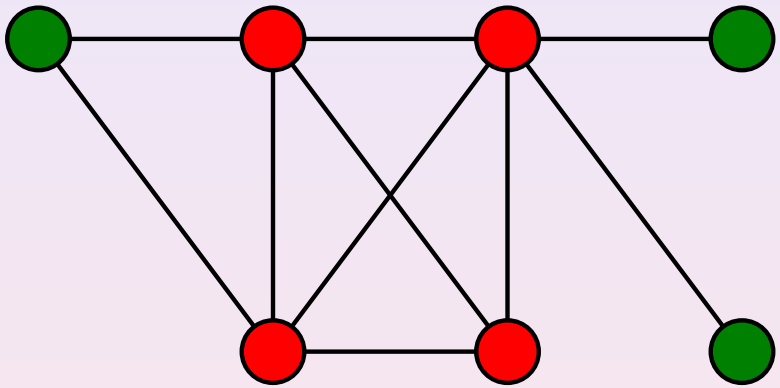
.

Clique: Dado un grafo $G = (V, E)$, un clique es un subconjunto maximal totalmente conectado. Es decir, un subconjunto $V_t \subseteq V$ tal que $\forall v_1, v_2 \in V_t, \quad (v_1, v_2) \in E$, y que no está estrictamente incluido en otro conjunto que cumpla esta propiedad.
El problema del Clique Máximo (Clique): Dado un grafo $G = (V, E)$ y un número natural $J \leq |V|$, determinar si existe un clique de tamaño mayor o igual que J.

Este problema es equivalente a la existencia de un conjunto totalmente conectado de tamaño mayor o igual que J.

Ejemplo

En el siguiente grafo, tenemos un clique de tamaño 4 (en rojo):



Cubrimiento por Vértices (CV)

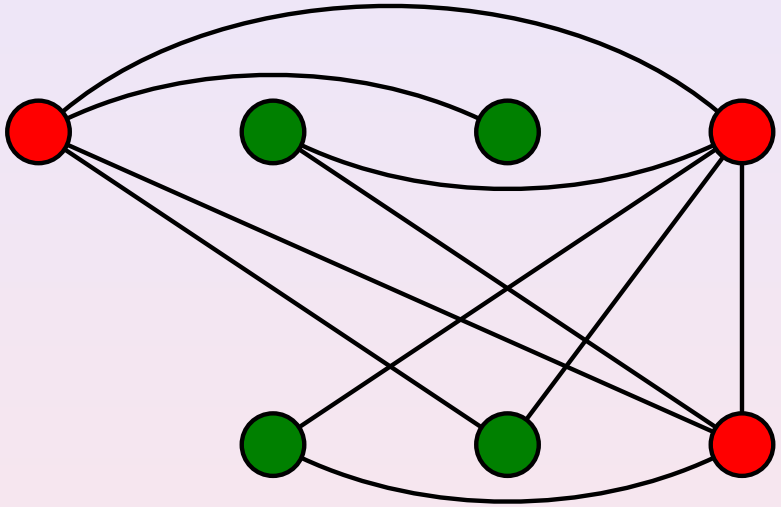
Dado un grafo $G = (V, E)$ y un subconjunto $V_c \subseteq V$, se dice que V_c es un cubrimiento por vértices de G, si y solo si toda arista del grafo tiene un extremo en V_c:

$$\forall (u, v) \in E, \quad (u \in V_c \vee v \in V_c)$$

Problema: Dado un grafo $G = (V, E)$ y un número natural $K \leq |V|$, determinar si existe un cubrimiento por vértices de tamaño menor o igual que K.

Ejemplo

En el siguiente grafo, tenemos un cubrimiento por vértices de tamaño 3 (en rojo):



Conjunto Independiente (CI)

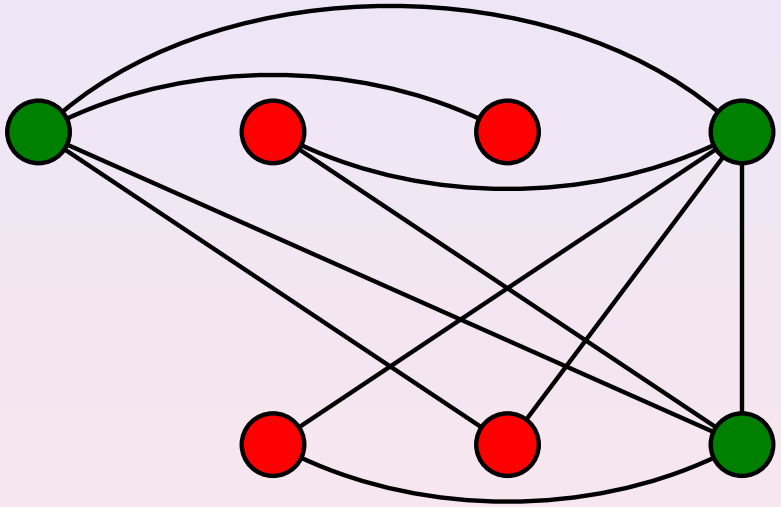
Dado un grafo $G = (V, E)$ y un subconjunto $V_i \subseteq V$, se dice que V_i es un conjunto independiente de G, si y solo si no hay ninguna arista que una vértices de V_i:

$$\forall u, v \in V_i, \quad (u, v) \notin E$$

Problema: Dado un grafo $G = (V, E)$ y un número natural $J \leq |V|$, determinar si existe un conjunto independiente de tamaño mayor o igual que J.

Ejemplo

En el siguiente grafo, tenemos un conjunto independiente de tamaño 4 (en rojo):



Lema

Si $G = (V, E)$ es un grafo y $V^* \subseteq V$, entonces las siguientes condiciones son equivalentes:

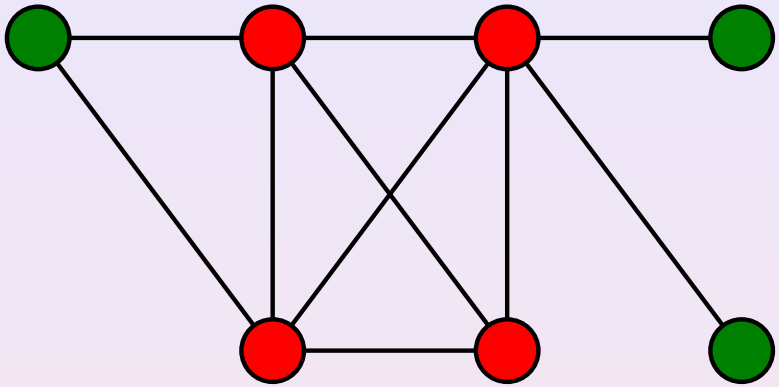
a) V^* es un cubrimiento por vértices de G

b) $V \setminus V^*$ es un conjunto independiente de G

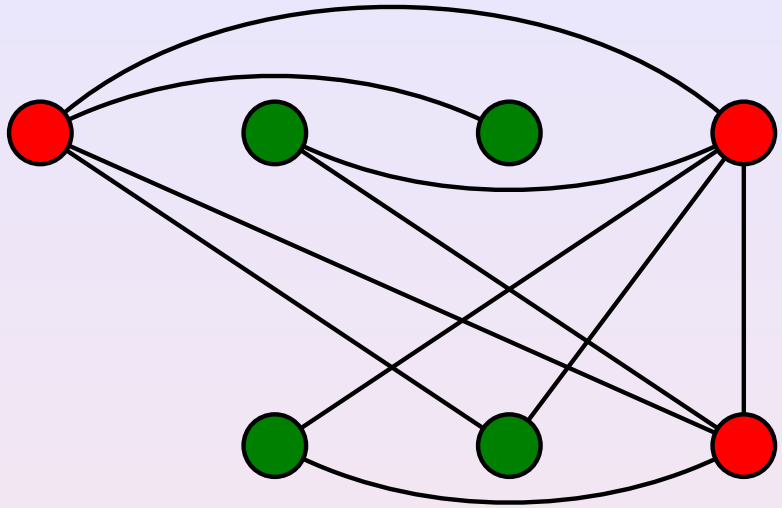
c) $V \setminus V^*$ es un subgrafo totalmente conectado del grafo complementario $\overline{G} = (V, \overline{E})$.
Donde $\overline{E} = V \times V \setminus E$.

Ejemplo

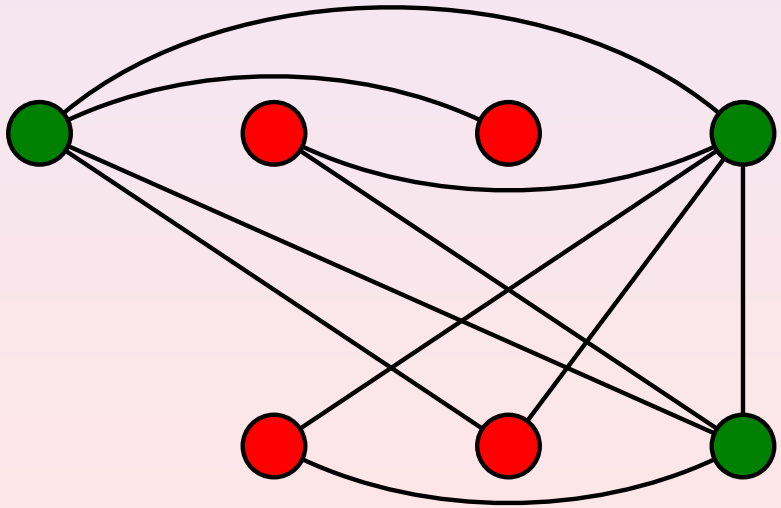
Totalmente Conectado



Cubrimiento por Vértices



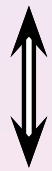
Conjunto Independiente



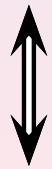
Equivalencia de Problemas

Los tres problemas (Clique, Cubrimiento por Vértices y Conjunto Independiente) son equivalente y si uno es NP-completo los otros también lo son.
Si n es es número de nodos de G, entonces:

G tiene un cubrimiento por vértices de tamaño menor o igual que K



G tiene un conjunto independiente de tamaño mayor o igual que $n-K$



$\overline{G}$ tiene un clique de tamaño mayor o igual que $n-K$

Cubr. por Vértices (CV) es NP-completo

Está claro que es NP: se elige de forma no determinista un subconjunto de vértices y se comprueba en tiempo polinómico si es un cubrimiento.

Reducción de 3-SAT a CV

Para demostrar que es completo reduciremos 3-SAT: $3\text{-SAT} \propto \text{CV}$.
Sea una ejemplo de 3-SAT con variables U y cláusulas C.
Se crea un ejemplo de CV, dado por un grafo $G = (V, E)$.

Elementos del grafo

- Para cada variables $u_i \in U$ se añaden dos vértices $u_i, \overline{u}_i$ y un arco que los una.
- Para cada cláusula $c_j \in C$, se añaden tres vértices $a_1[j], a_2[j], a_3[j]$ y tres aristas que los unan (se forma un triángulo).
- Para cada clásula $c_j \in C$ si en el literal número k de esta clásula aparece la variable u_i, se añade una arista de $u_k[j]$ al vértice u_i si la variable aparece positiva y al vértice $\overline{u}_i$ si la variable aparece negada.
- Se pone un límite $K = n + 2m$.

Ejemplo

Ejemplo de 3-SAT:

$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}.$

Ejemplo

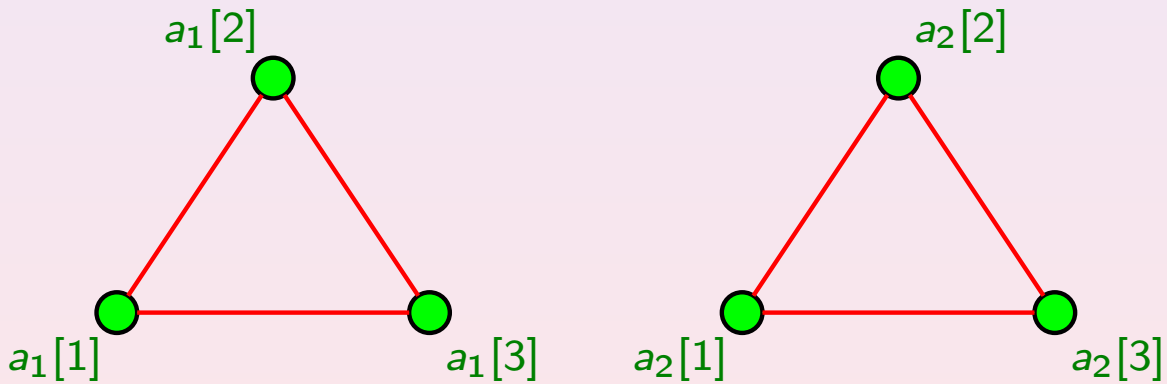
Ejemplo de 3-SAT:

$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}.$

Ejemplo

Ejemplo de 3-SAT:

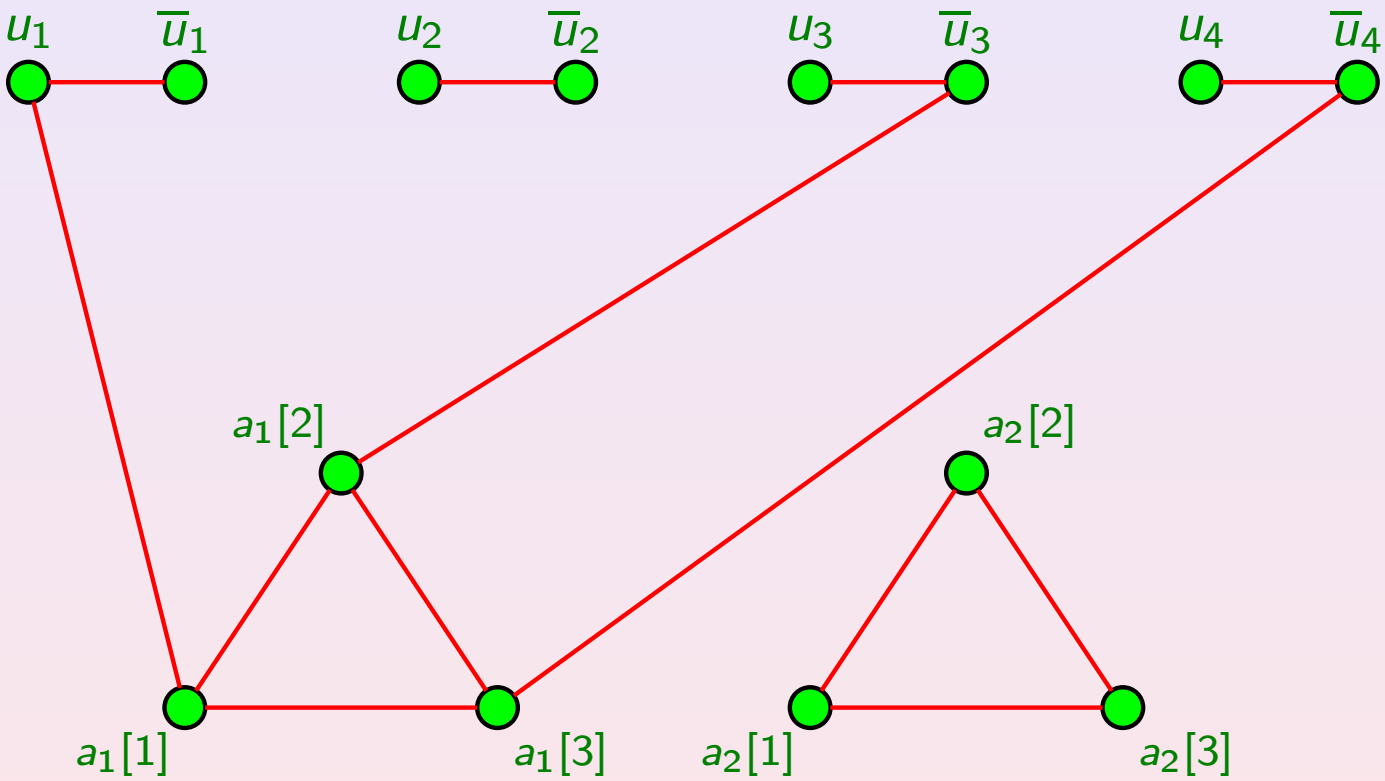
$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}.$



Ejemplo

Ejemplo de 3-SAT:

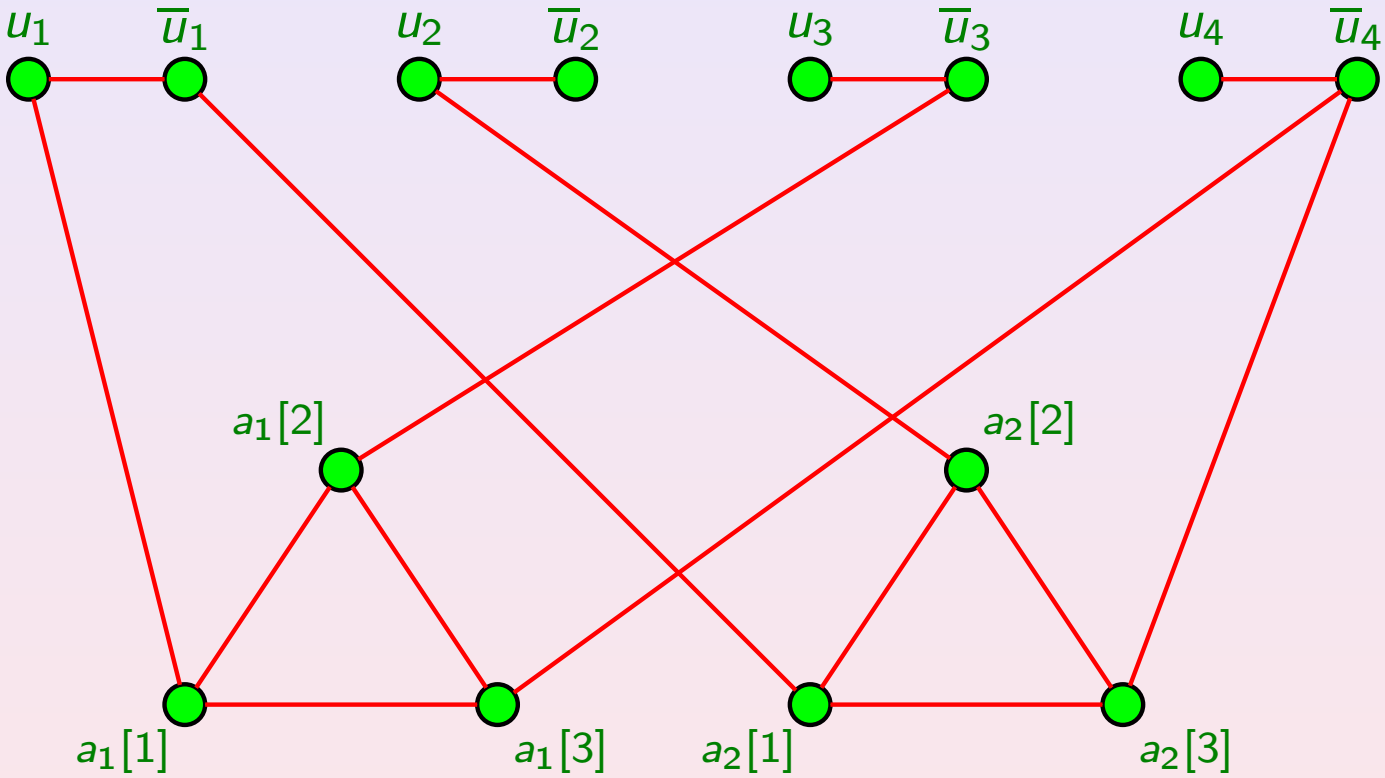
$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}.$



Ejemplo

Ejemplo de 3-SAT:

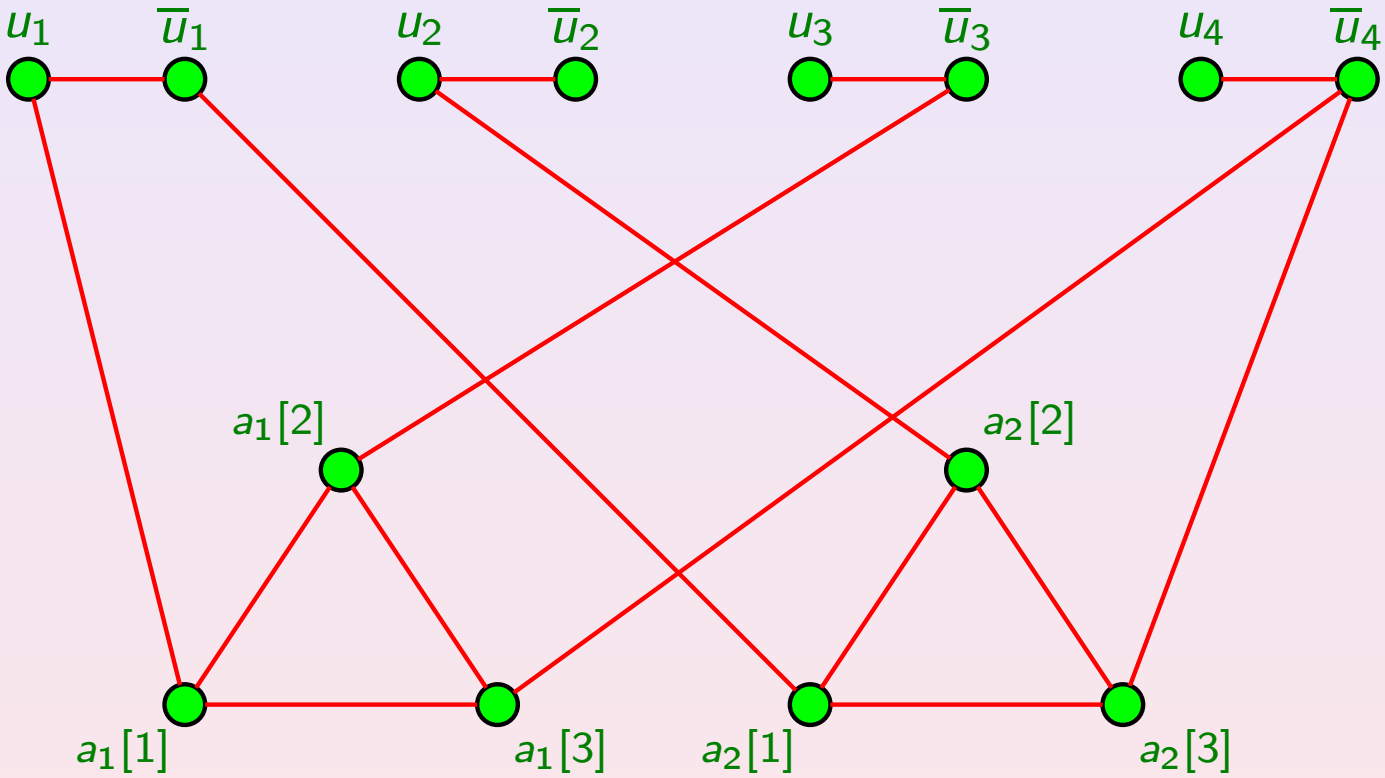
$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}.$



Ejemplo

Ejemplo de 3-SAT:

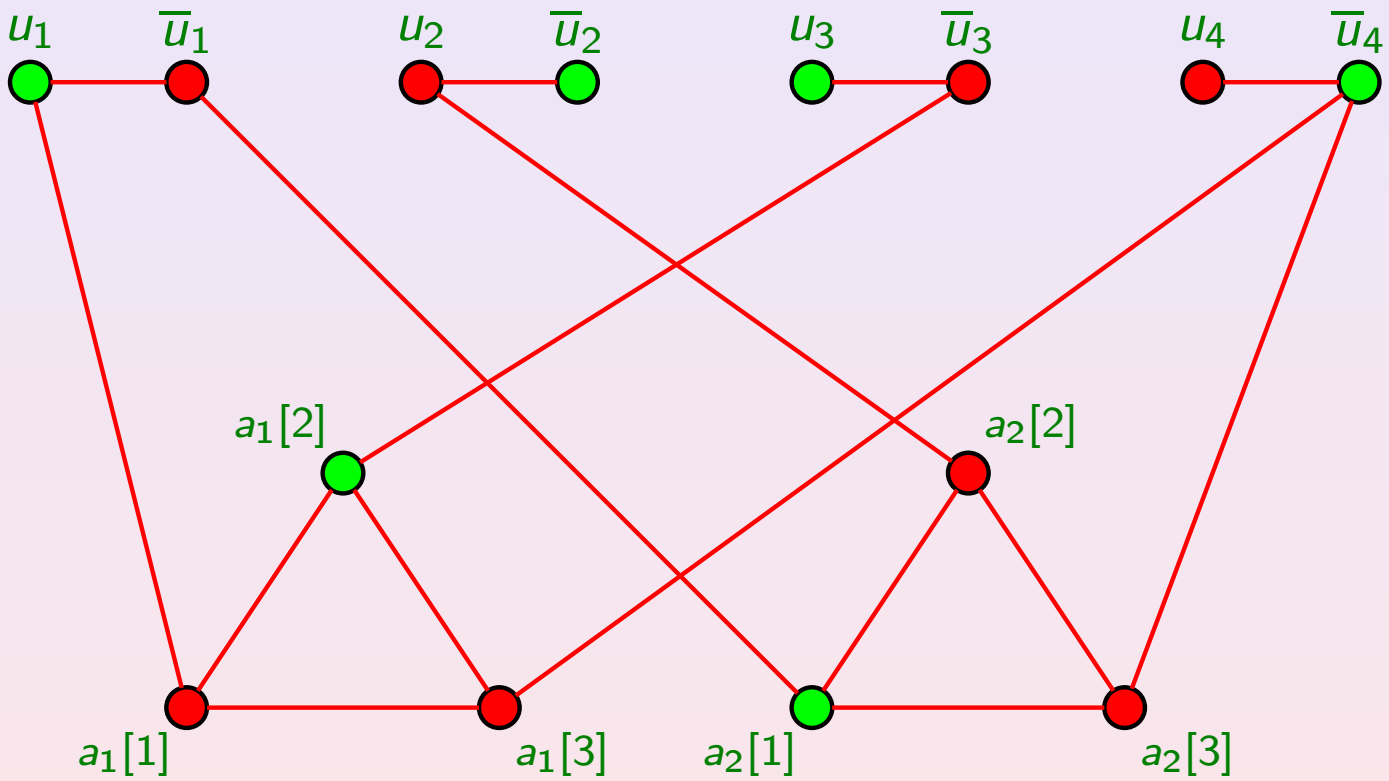
$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}.$



Equivalencia CV $\leftrightarrow$ 3-SAT

Ejemplo de 3-SAT:

$U = \{u_1, u_2, u_3, u_4\}, \quad C = \{u_1 \vee \neg u_3 \vee \neg u_4, \neg u_1 \vee u_2 \vee \neg u_4\}.$



u_1 falso

u_2 verdadero

u_3 falso

u_4 verdadero

Cubrimiento:

Vértices rojos

CV implica 3-SAT

Si existe un cubrimiento V_c, entonces

- Debe de haber, al menos, un vértice de cada pareja, $u_i, \overline{u_i}$
- Para cada cláusula C_j deben de existir, al menos, dos vértice de cada conjunto: $\{a_1[j], a_2[j], a_3[j]\}$.

Como el cubrimiento tiene, a lo más, $n+2m$ vértices, entonces habrá exactamente un vértice por cada pareja $u_i, \overline{u_i}$ y dos por cada conjunto $\{a_1[j], a_2[j], a_3[j]\}$.

CV implica 3-SAT

Las clásulas se pueden satisfacer haciendo para cada variable u_i:

- u_i cierto si $u_i \in V_c$
- u_i falso si $u_i \notin V_c$

Cada cláusula C_j tiene tres vértices, y cada vértice está conectado con un vértice de variable.
De estos tres arcos, hay dos que tienen extremos en los dos vértices de $\{a_1[j], a_2[j], a_3[j]\}$ que están en V_c
El otro arco, tendrá que tener su extremo del cubrimiento en los vértices correspondientes a las variables: u_i o $\overline{u_i}$. El valor de verdad asignado a dicha variable hace que esa cláusula se satisfaga.

SAT implica CV

Sea una asignación de valores de verdad que haga consistentes las cláusulas, entonces un cubrimiento por vértices del tamaño deseado se consigue de la siguiente forma:

$$\begin{aligned} V_c = \quad & \{u_i : u_i \text{es verdadero}\} \cup \{\overline{u_i} : u_i \text{es falso}\} \cup \\ & \left(\bigcup_j (\{a_1[j], a_2[j], a_3[j]\} - \{a_i[j]\})\right) \end{aligned}$$

donde $a_i[j]$ corresponde al literal que hace verdadera la cláusula C_j.

El Problema del Circuito Hamiltoniano (CH)

El problema del Circuito Hamiltoniano es NP-completo.

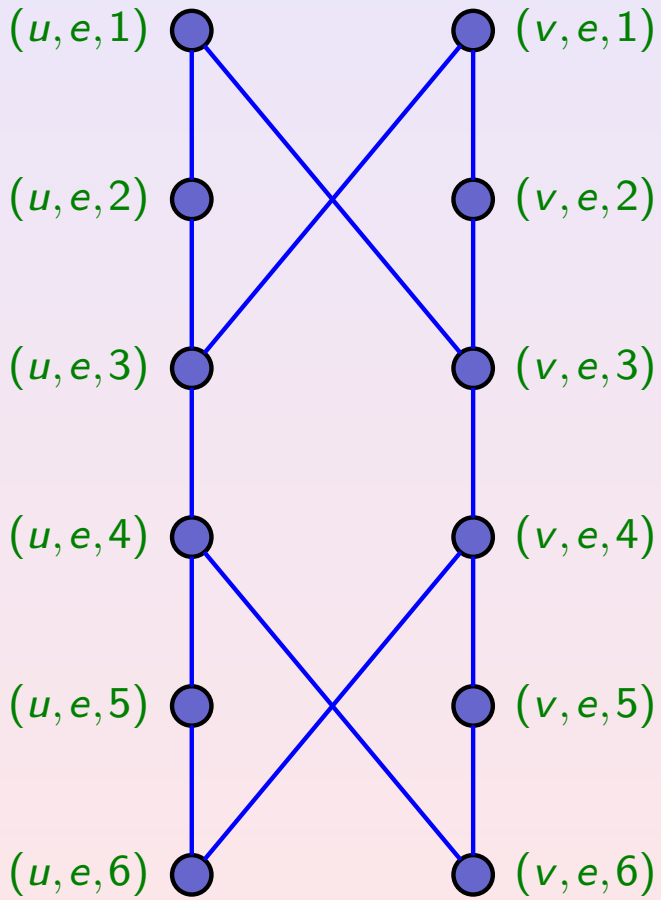
Es inmediato que es NP. El algoritmo no-determinista polinomico solo tiene que elegir n nodos (número de nodos en el grafo) y después comprobar que hay un arco desde cada nodo al siguiente y del último hasta el primero.

Para demostrar que es completo, vamos a reducir CV a este problema. Sea $G = (V, E)$ y $K \leq |V|$ un problema de cubrimiento por vértices.

Vamos a construir un grafo $G' = (V', E')$ de tal forma que la existencia de un circuito hamiltoniano para G' sea equivalente a la existencia de un recubrimiento de tamaño $\leq K$ para G.

Reducción: grafos base

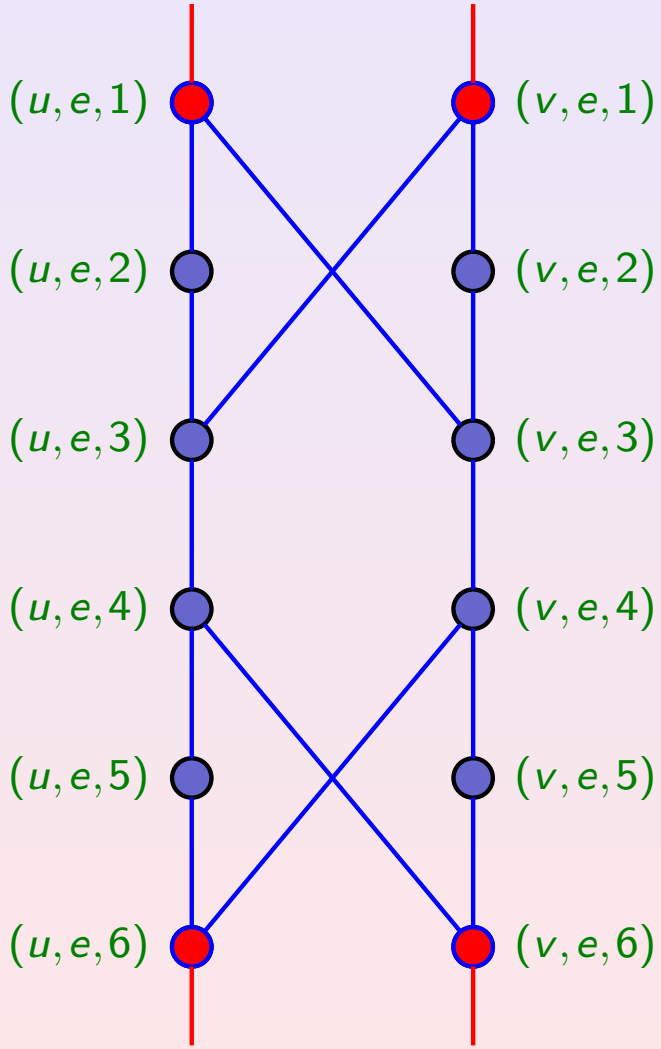
Para cada $e = (u, v) \in E$ se añade el siguiente grafo a G' (con 12 vértices).



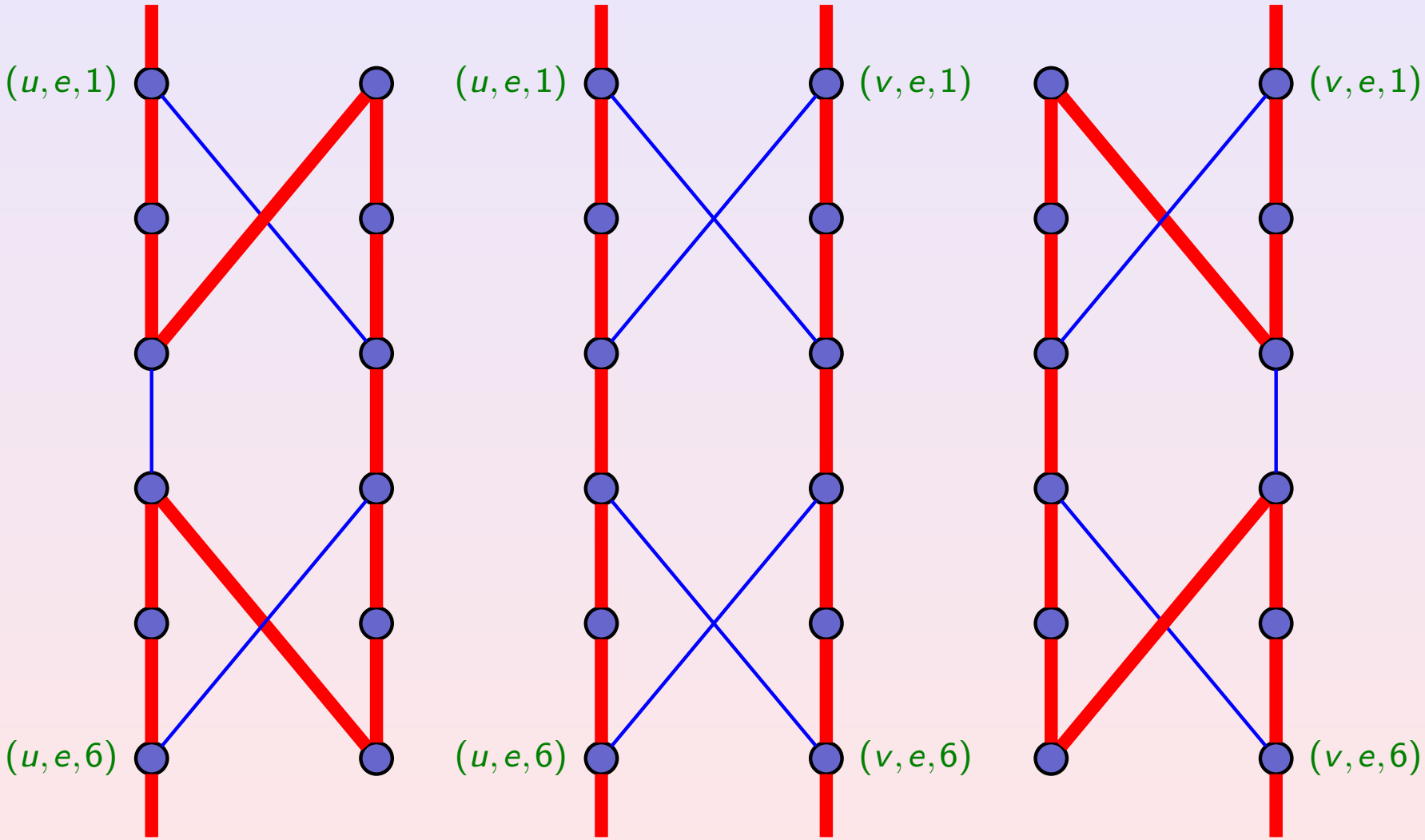
Reducción: grafos base

Para cada $e = (u, v) \in E$ se añade el siguiente grafo a G' (con 12 vértices).

Solo estos nodos de los extremos se conectan con el resto del grafo



Reducción: recorridos

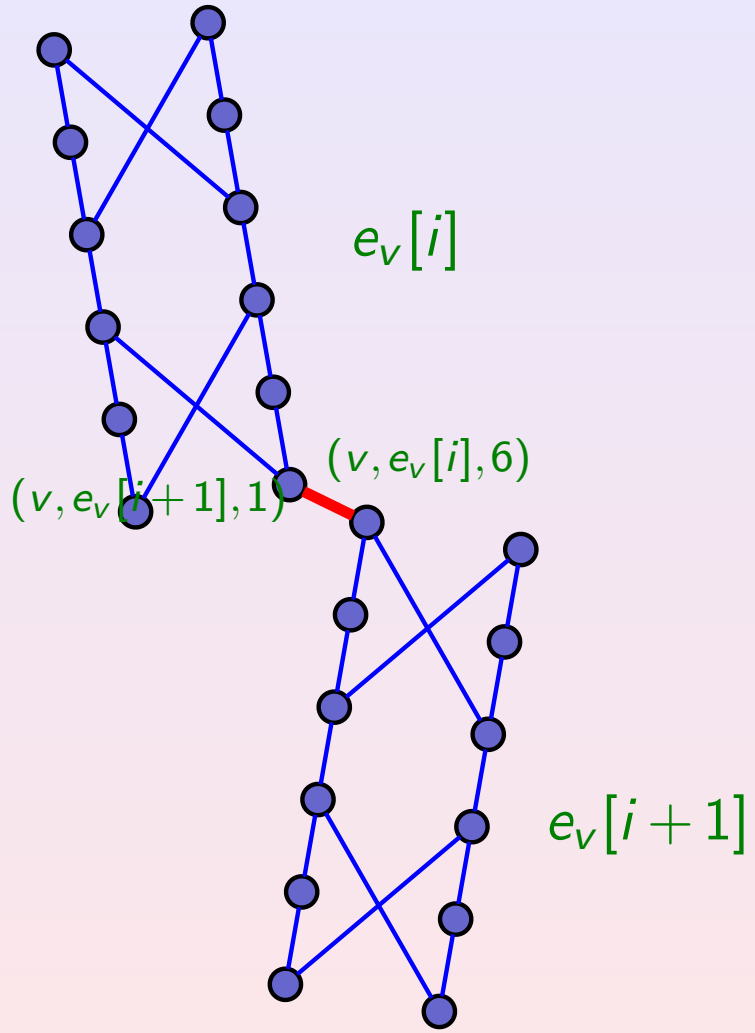


Reducción: conectando grafos básicos

Para cada $v \in V$ sea

$$e_v[1], \ldots, e_v[r_v]$$

una ordenación arbitraria de los arcos que contienen v.
Para $1 \leq i < r_v$ unimos el subgrafo asociado a $e_v[i]$ y el subgrafo $e_v[i+1]$ mediante un arco que va de $(v, e_v[i], 6)$ a $(v, e_v[i+1], 1)$.

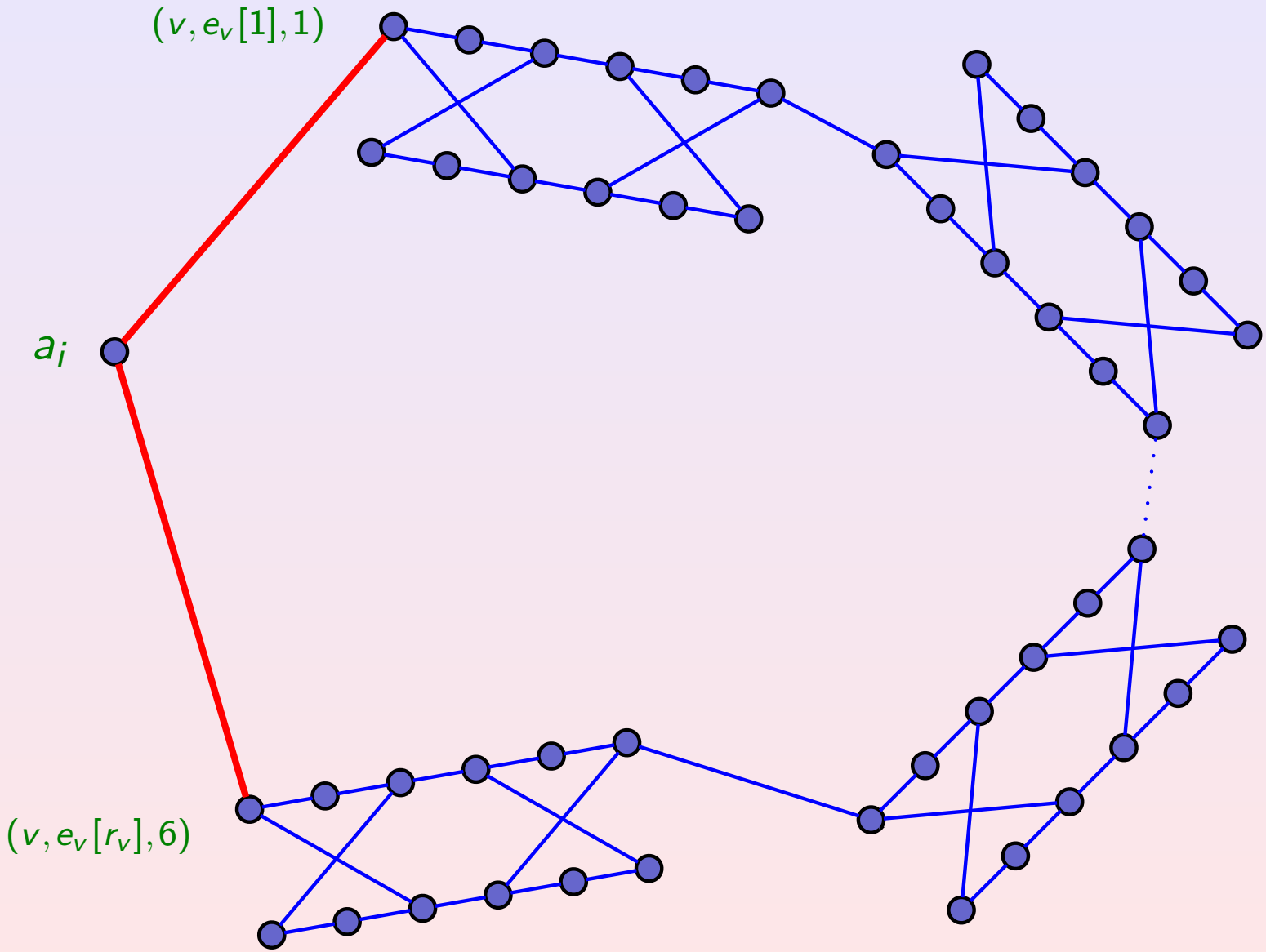


Reducción: Nodos Adicionales

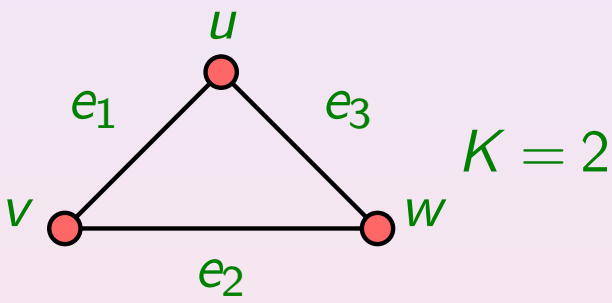
Finalmente se añaden K vértices, $a_1, a_2, \ldots, a_K$, que se unen con los subgrafos de la siguiente forma:

- Para cada $v \in V$ sea $e_v[1], \ldots, e_v[r_v]$, la lista de las aristas que lo contienen en el orden que se consideraron anteriormente
  - Se añade una arista de cada uno de los a_i a $(v, e_v[1], 1)$ (vértice extremo de la primera arista correspondiente a v)
  - Se añade una arista de cada uno de los a_i a $(v, e_v[r_v], 6)$ (vértice extremo de la última arista correspondiente a v)

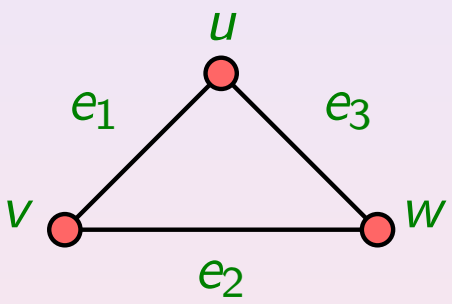
Reducción: Nodos Adicionales



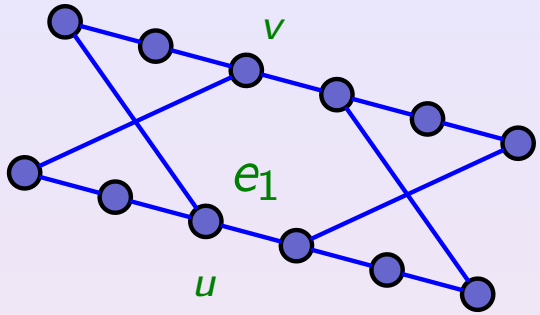
Ejemplo

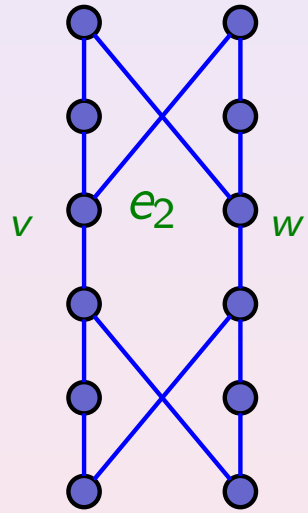


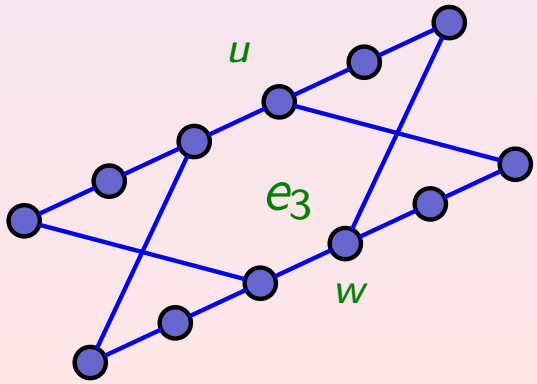
Ejemplo



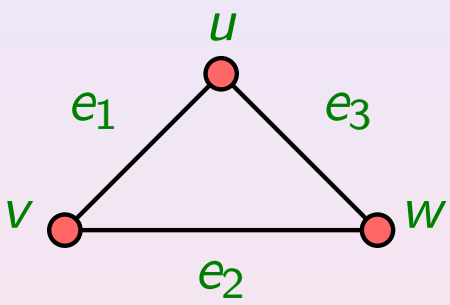
$K = 2$







Ejemplo

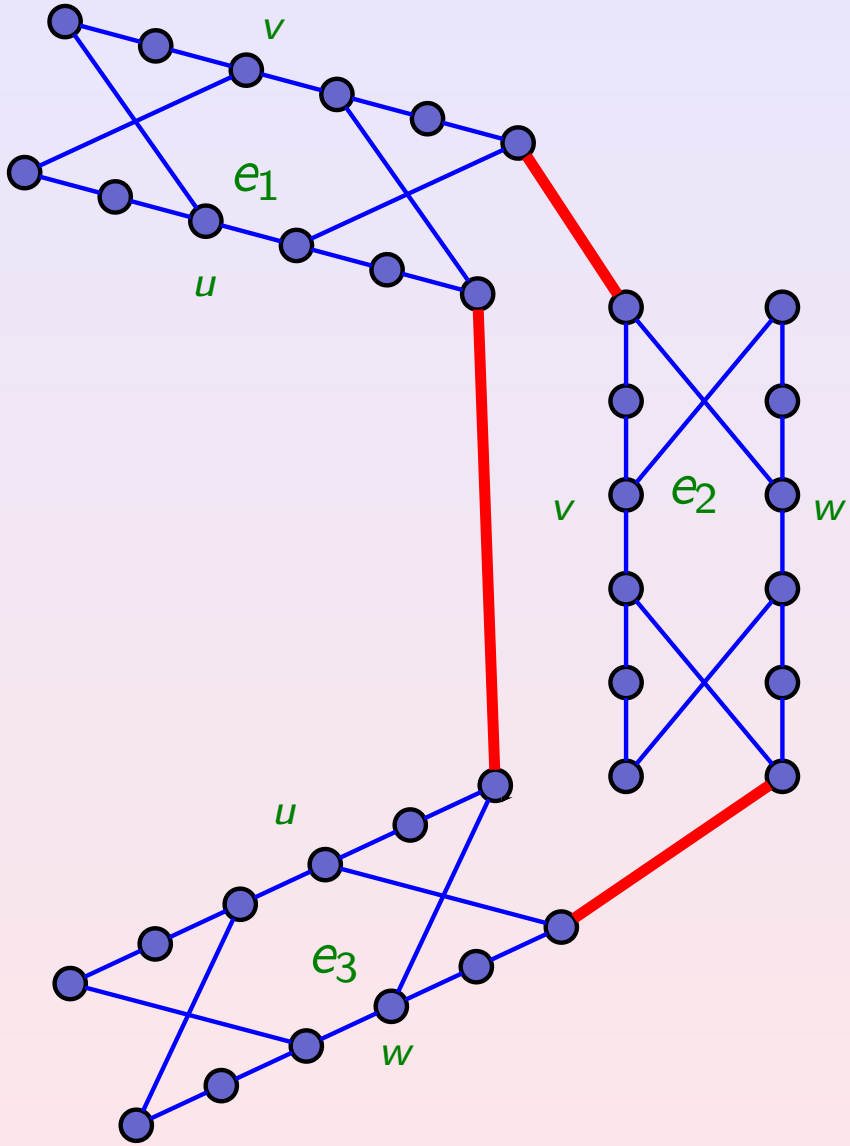


$K = 2$

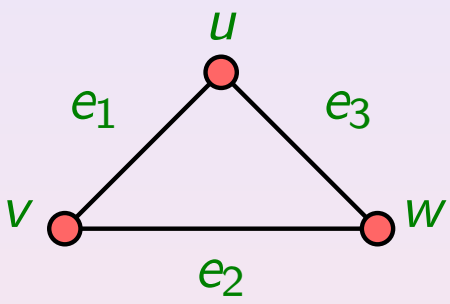
Nodo u: aristas e_1, e_3

Nodo v: aristas e_1, e_2

Nodo w: aristas e_2, e_3



Ejemplo



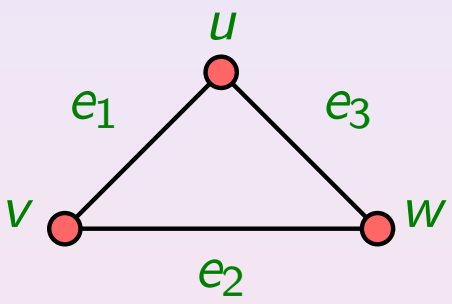
$K = 2$

Nodo u: aristas e_1, e_3

Nodo v: aristas e_1, e_2

Nodo w: aristas e_2, e_3

Ejemplo

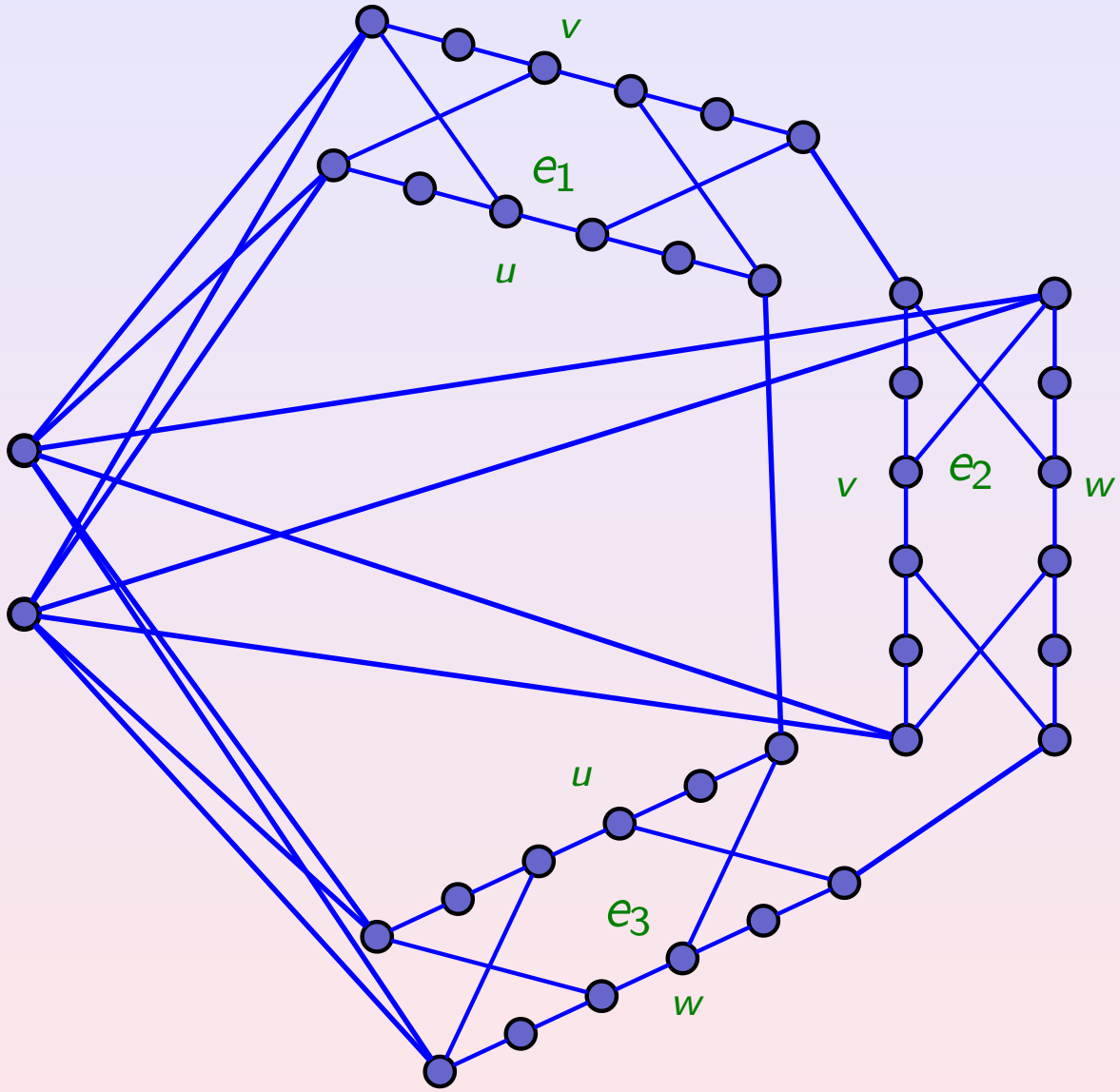


Nodo u: aristas e_1, e_3

Nodo v: aristas e_1, e_2

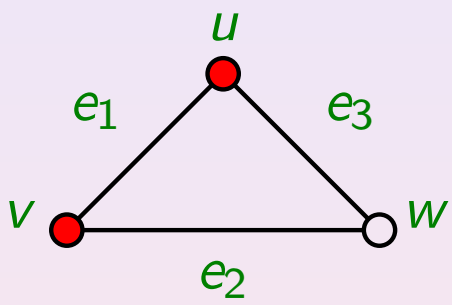
Nodo w: aristas e_2, e_3

$K = 2$



Equivalencia de Soluciones

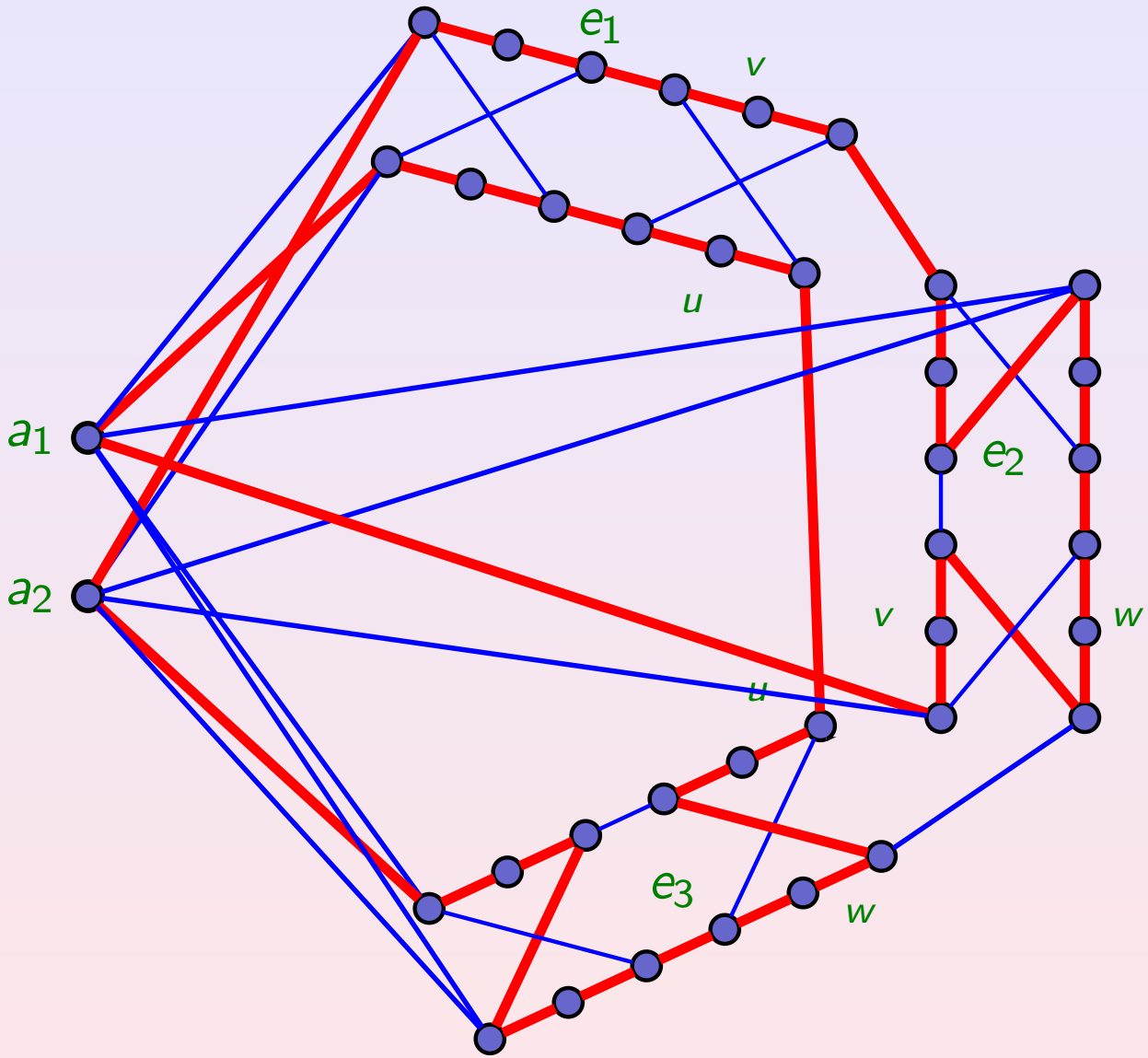
$K = 2$



Nodo u: aristas e_1, e_3

Nodo v: aristas e_1, e_2

Nodo w: aristas e_2, e_3



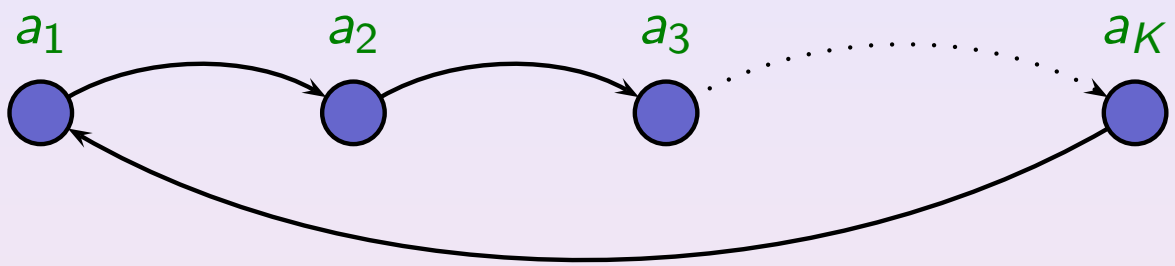
Equivalencia de Soluciones: CV $\Rightarrow$ CH

Sean K vértices que forman un cubrimiento por vértices: $\{v_1, \ldots, v_K\}$. Un circuito hamiltoniano se puede construir de la siguiente forma comenzando en a_1:

- Si estamos en a_i, desde él recorremos todos los grafos asociados a las aristas de v_i: para cada arista si tiene sólo a v_i en el cubrimiento por vértices, se recorre el subgrafo de esa arista de forma completa; si la arista contiene los dos extremos en el cubrimiento, se recorre sólo la mitad de los vértices del subgrafo correspondientes a v_i. Desde el último subgrafo volvemos a a_{i+1}, repitiendo el proceso, excepto para a_K que volvemos a a_1 y termina el circuito.

Equivalencia de Soluciones: CH $\Rightarrow$ CV

Si tenemos un circuito hamiltoniano, tendrá la forma



Desde cada a_i al siguiente a_{i+1} (y también desde a_K a a_1) recorremos los subgrafos. En cada uno de esos recorridos entramos y salimos en los subgrafos por un mismo vértice. Sea este vértice v_i (v_K si es el recorrido de a_K a a_1).

El conjunto $\{v_1, \ldots, v_K\}$ es un cubrimiento por vértices, ya que cada arista tiene un subgrafo en el que, al menos, el circuito hamiltoniano entra una vez. El vértice por el que se entra (que ha de coincidir con el de salida) ha de estar en el cubrimiento por vértices.

El Problema de la SUMA

Datos: Un conjunto A y un tamaño para cada uno de sus elementos:

$$s : A \to \mathbb{N}$$

y un número entero B **Pregunta:** Determinar si existe un $A' \subseteq A$ tal que se verifique:

$$\sum_{a \in A'} s(a) = B$$

Este es claramente un problema de NP: se eligen de forma no determinista los elementos de A' y en tiempo polinómico se determina si la suma de los tamaños de los conjuntos es igual a B.
Para demostrar que es completo para NP, vamos a reducir el cubrimiento por tripletas (ACTRI) a este problema.

Reducción, ACTRI $\propto$ SUMA

Sea W, X, Y con $|W| = |X| = |Y| = q$ y un subconjunto $M \subseteq X \times Y \times Z$ un ejemplo del problema ACTRI, vamos a construir una partición equivalente.
Consideremos:

$$W = \{w_1, \ldots, w_q\}, \qquad X = \{x_1, \ldots, x_q\}$$
$$Y = \{y_1, \ldots, y_q\}$$
$$M = \{m_1, m_2, \ldots, m_k\}$$

El conjunto A va a contener k elementos $A = \{a_1, \ldots, a_k\}$.
Cada elemento $a_i \in A$ se corresponde con una tripleta $m_i \in M$.

Cuestión Previa

Pregunta

Si sumanos k unos, ¿cuantas cifras como máximo puede tener el número resultante?

Cuestión Previa

Pregunta

Si sumanos k unos, ¿cuantas cifras como máximo puede tener el número resultante?

$$\begin{array}{ccccc} & & & & 1 \\ & & & & 1 \\ & & & & 1 \\ & & \cdots & & \\ & & & & 1 \\ \hline 1 & 0 & 1 & 1 & 1 \end{array}$$

El resultado tiene a lo más tamaño $p = [\log_2(k)] + 1$ (sumar uno a $\log_2(k)$ sin decimales).

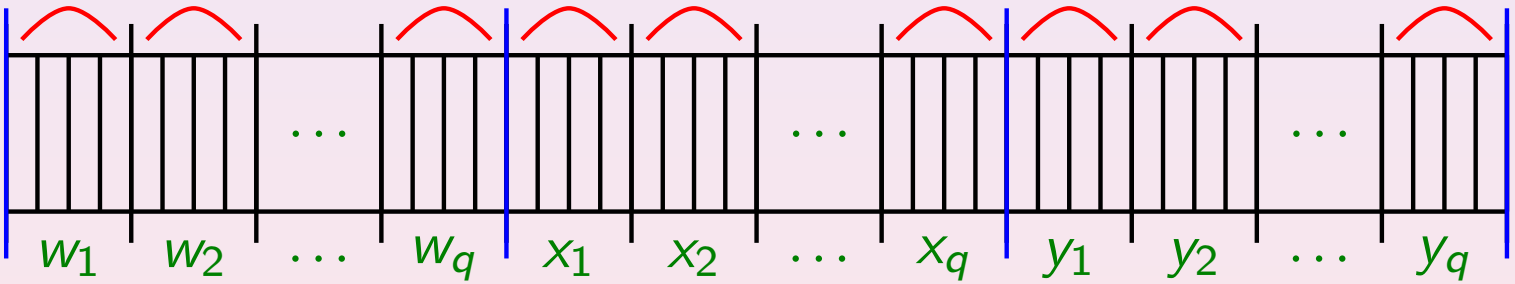
Reducción ACTRI $\propto$ SUMA

Para cada tripleta $m_i = (w_{f(i)}, x_{g(i)}, y_{h(i)})$, se considera el elemento a_i con un peso:

$$s(a_i) = 2^{p(3q-f(i))} + 2^{p(2q-g(i))} + 2^{p(q-h(i))}$$

donde $p = [\log_2(k)] + 1$.

En binario podemos ver el número en grupos de p posiciones. Cada grupo corresponde a un elemento de W, X o Y:

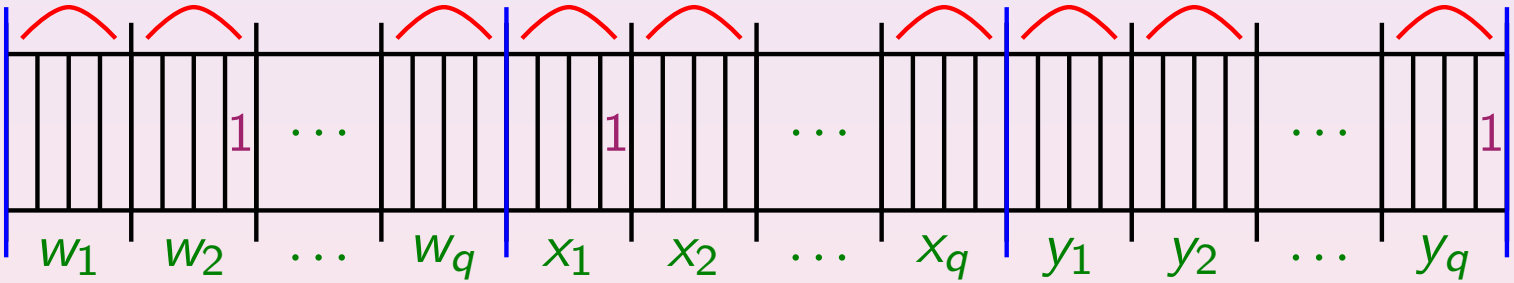


Cada tripleta $m_i = (w_{f(i)}, x_{g(i)}, y_{h(i)})$ se corresponde con un número con un 1 a la derecha de las zonas de $w_{f(i)}, x_{g(i)}, y_{h(i)}$ y 0 en el resto.

Para cada tripleta $m_i = (w_{f(i)}, x_{g(i)}, y_{h(i)})$, se considera el elemento a_i con un peso:

$$s(a_i) = 2^{p(3q-f(i))} + 2^{p(2q-g(i))} + 2^{p(q-h(i))}$$

donde $p = [\log_2(k)] + 1$.
En binario podemos ver el número en grupos de p posiciones. Cada grupo corresponde a un elemento de W, X o Y:



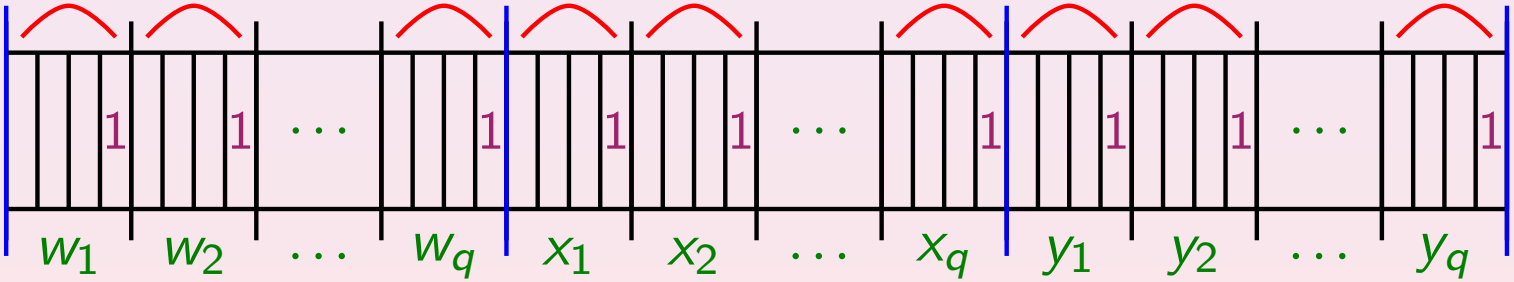
Cada tripleta $m_i = (w_{f(i)}, x_{g(i)}, y_{h(i)})$ se corresponde con un número con un 1 a la derecha de las zonas de $w_{f(i)}, x_{g(i)}, y_{h(i)}$ y 0 en el resto.
Ejemplo: Tripleta: (w_2, x_1, y_q)

Reducción ACTRI $\propto$ SUMA

Si $p = [\log_2(k)] + 1$, entonces sumando k o menos unos nunca obtenemos un número con un 1 más allá de la posición $p+1$

Sumando k o menos números de los asociados a las tripletas, nunca pasa un 1 de la zona de un elemento a la zona de otro.

Finalmente, $B = \sum_{j=0}^{3q-1} 2^{p \cdot j}$. Este número tiene un 1 a la derecha de cada una de las zonas:



Ejemplo

Supongamos el problema de tripletas con:
$W = \{w_1, w_2, w_3\}, X = \{x_1, x_2, x_3\}, Y = \{y_1, y_2, y_3\}$ y las tripletas
$(w_2, x_1, y_3), (w_2, x_2, y_2), (w_3, x_2, y_1), (w_3, x_3, y_2), (w_1, x_3, y_2)$.

Ejemplo

Supongamos el problema de tripletas con:
$W = \{w_1, w_2, w_3\}, X = \{x_1, x_2, x_3\}, Y = \{y_1, y_2, y_3\}$ y las tripletas $(w_2, x_1, y_3), (w_2, x_2, y_2), (w_3, x_2, y_1), (w_3, x_3, y_2), (w_1, x_3, y_2)$.
Construimos el siguiente problema de la SUMA: el número de tripletas es $k = 5$, y $p = [\log(k)] + 1 = 3$. Hay un individuo para cada tripleta definido por su peso (se supone un orden $w_1, w_2, w_3, x_1, x_2, x_3, y_1, y_2, y_3$):

	w_1	w_2	w_3	x_1	x_2	x_3	y_1	y_2	y_3	
1	000	001	000	001	000	000	000	000	001	(w_2, x_1, y_3)
2	000	001	000	000	001	000	000	001	000	(w_2, x_2, y_2)
3	000	000	001	000	001	000	001	000	000	(w_3, x_2, y_1)
4	000	000	001	000	000	001	000	001	000	(w_3, x_3, y_2)
5	001	000	000	000	000	001	000	001	000	(w_1, x_3, y_2)

y $B = 001\ 001\ 001\ 001\ 001\ 001\ 001\ 001\ 001$.

Equivalencia de las soluciones

Las tripletas 1, 3, 5: $(w_2, x_1, y_3), (w_3, x_2, y_1), (w_1, x_3, y_2)$ son una solución del problema original y los individuos correspondientes a estas tripletas suman B.

1	000 001 000 001 000 000 000 000 001	(w_2, x_1, y_3)
3	000 000 001 000 001 000 001 000 000	(w_3, x_2, y_1)
5	001 000 000 000 000 001 000 001 000	(w_1, x_3, y_2)
B	001 001 001 001 001 001 001 001 001	

Reducción ACTRI $\propto$ SUMA

Como una consencuencia, para cada $A' \subseteq \{a_1, \ldots, a_k\}$, tenemos que

$$\sum_{a \in A'} s(a) = B$$

si y solo si $M' = \{m_i : a_i \in A'\}$ es un recubrimiento por tripletas.

El Problema de la Partición

Datos: Un conjunto C y un tamaño para cada uno de sus elementos:

$$s : C \to \mathbb{N}$$

Pregunta: Determinar si existe un $C' \subseteq C$ tal que se verifique:

$$\sum_{a \in C'} s(a) = \sum_{a \in C \setminus C'} s(a)$$

Este es claramente un problema de NP: se eligen de forma no determinista los elementos de C' y en tiempo polinómico se determina si los tamaños totales de los conjuntos C' y $C \setminus C'$ son iguales.

Para demostrar que es NP-completo, vamos a reducir el probmea SUMA a este problema.

Reducción SUMA a PARTICIÓN

Si tenemos un problema de suma con A, tamaños s y entero B, entonces creamos un problema de la partición en el que $C = A \cup \{b_1, b_2\}$ (añadimos dos valores nuevos).
Los tamaños en el problema de la PARTICIÓN de los elementos de A son los mismos que en el problema de la SUMA.
Los tamaños de los nuevos elementos, b_1 y b_2, son:

$$s(b_1) = \left(\sum_{i=1}^{k} s(a_i)\right)$$
$$s(b_2) = 2B$$

Cada uno de estos pesos necesita, a lo más, $(3pq)$ bits. Se pueden calcular zona a zona de forma consecutiva.

Conjunto C

El conjunto C es igual a $\{a_1, \ldots, a_k, b_1, b_2\}$.

La suma de los pesos de sus elementos es

$$\sum_{x \in A} s(x) = \sum_{i=1}^{k} s(a_i) + s(b_1) + s(b_2) =$$

$$\sum_{i=1}^{k} s(a_i) + \sum_{i=1}^{k} s(a_i) + 2B = 2 \sum_{i=1}^{k} s(a_i) + 2B$$

Este conjunto se parte en dos mitades cuando cada una pesa:

$$\sum_{i=1}^{k} s(a_i) + B$$

Equivalencia de soluciones

Si la solución es positiva al problema de la PARTICIóN, entonces es positiva para la SUMA

Supongamos $C' \subseteq C$, tal que

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$$

Entonces cada una de estas sumas es $\sum_{i=1}^{k} s(a_i) + B$. Uno de los conjuntos (supongamos que es A') debe de contener b_1 y no b_2.
La suma de los pesos de los elementos de C' distintos de b_1 tiene que ser B.
Por tanto, existe una solución positiva al problema de la SUMA.

Equivalencia de soluciones

Si la solución es positiva a SUMA, entonces es positiva al problema de la PARTICIÓN

Si el problema de la suma tiene solución con A' tal que $\sum_{a \in A'} = B$, entonces $C' = \{b_1\} \cup A'$ ya que la suma de los pesos de C' es

$$(\sum_{i=1}^{k} s(a_i) + B)$$

que es la mitad del total.

Técnicas de Reducción

Del libro de Garey-Johnson:

- Restricción. Demostrando que un problema NP-completo es un subproblema del que estamos considerando.
- Reemplazamiento Local. Haciendo una transformación de un problema NP-completo elemento a elemento.
  - Reemplazamiento Local con Refuerzo. Haciendo una transformación de un problema NP-completo elemento a elemento, pero añadiendo algunos elementos adicionales para forzar la equivalencia de los problemas.
- Diseño de Componentes. Distintos elementos del problema original se transforman en distintos tipos de estructura que se conectan con otros elementos.

Técnica de la Restricción

Regla

Si un problema Π_1 es NP y un subproblema suyo, Π_2, es NP-completo, entonces Π_1 es NP-completo.

'Subproblema' significa que eligiendo unos parámetros concretos de Π_2, obtenemos Π_1.

Ejemplo: Problema de la Mochila

Tenemos un conjunto finito de objetos U. Cada objeto, u, tiene un tamaño, $s(u) \in \mathbb{N}$, y un valor $v(u) \in \mathbb{N}$. Tenemos, además, dos números naturales: B (el tamaño máximo) y K (el valor mínimo). La pregunta es si existe un subconjunto de objetos $U' \subseteq U$, tal que

$$\sum_{u \in U'} s(u) \leq B, \qquad \sum_{u \in A'} v(u) \geq K$$

El problema de la partición es un caso particular de este problema, en el que $s(u) = v(u), \forall u$ y $B = K = (1/2) \sum_{u \in U} s(u)$.

Ejemplo: Problema de la Mochila

Tenemos un conjunto finito de objetos U. Cada objeto, u, tiene un tamaño, $s(u) \in \mathbb{N}$, y un valor $v(u) \in \mathbb{N}$. Tenemos, además, dos números naturales: B (el tamaño máximo) y K (el valor mínimo). La pregunta es si existe un subconjunto de objetos $U' \subseteq U$, tal que

$$\sum_{u \in U'} s(u) \leq B, \qquad \sum_{u \in A'} v(u) \geq K$$

El problema de la partición es un caso particular de este problema, en el que $s(u) = v(u), \forall u$ y $B = K = (1/2)\sum_{u \in U} s(u)$.

REDUCCIÓN de PARTICIÓN(C,s) a MOCHILA(U,s',v',B,K):
$U = C, s' = s, v' = s, B = K = (1/2)\sum_{u \in C} s(u)$

Asignación en un Multiprocesador

Datos: Un conjunto A de tareas, cada tarea, $a \in A$, tiene una longitud $l(a) \in \mathbb{N}$. Tenemos, además, un número de procesadores m y un tiempo límite, $D \in \mathbb{N}$.
Pregunta: ¿Existe una partición de A: $\{A_1, \ldots, A_m\}$ en m subconjuntos disjuntos, de manera que

$$\max \{ \sum_{a \in A_i} l(a) : 1 \leq i \leq m \} \leq D$$

Si nos restringimos al caso $m = 2$ y $D = 1/2 \sum_{a \in A} l(a)$ obtenemos el problema de la partición.

Reemplazamiento Local

Cada elemento de un problema NP-completo Π' se transforma en una estructura del problema que estamos considerando Π.

Ejemplo: Reducción de SAT a 3-SAT

Partición en Triángulos (PARTRI)

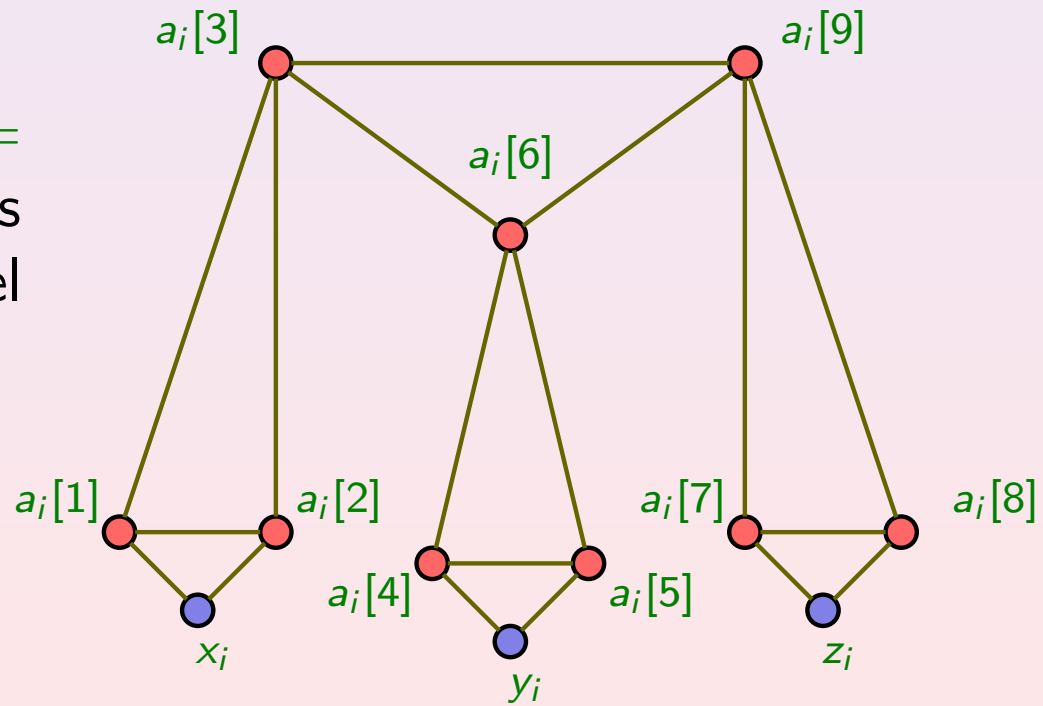
Se parte de un grafo $G = (V, E)$ con un número de vértices $|V| = 3q$.
La pregunta es si existe un partición de V en q conjuntos disjuntos de tamaño 3: $V_1, \ldots, V_q$, de tal manera que si $V_i = \{V_{i[1]}, V_{i[2]}, V_{i[3]}\}$, entonces los arcos $(V_{i[1]}, V_{i[2]}), (V_{i[2]}, V_{i[3]}), (V_{i[1]}, V_{i[3]}) \in E$ (los subconjuntos de vértices son triángulos).

Este es un problema de NP, y se puede demostrar que es completo reduciento 3-SET (cubrimiento por conjuntos de tamaño 3).

Reducción: 3-SET $\propto$ PARTRI

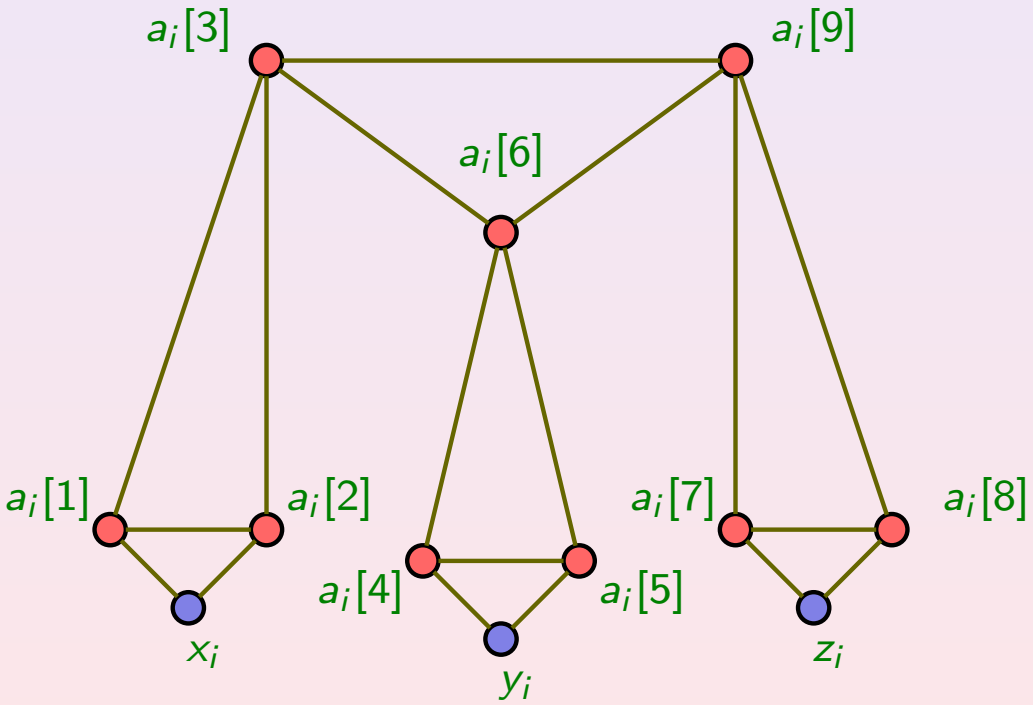
Sea X con $|X| = 3q$ y C una familia de subconjuntos de tres elementos de X.
Construimos el grafo $G = (V, E)$ con un número de vértices $|V| = 3q'$ de la siguinete forma: todos los elementos de X serán vértices del grafo.

Para cada conjunto $c_i = \{x_i, y_i, z_i\} \in C$, añadimos 9 nuevos vértices a V y el siguiente subgrafo:



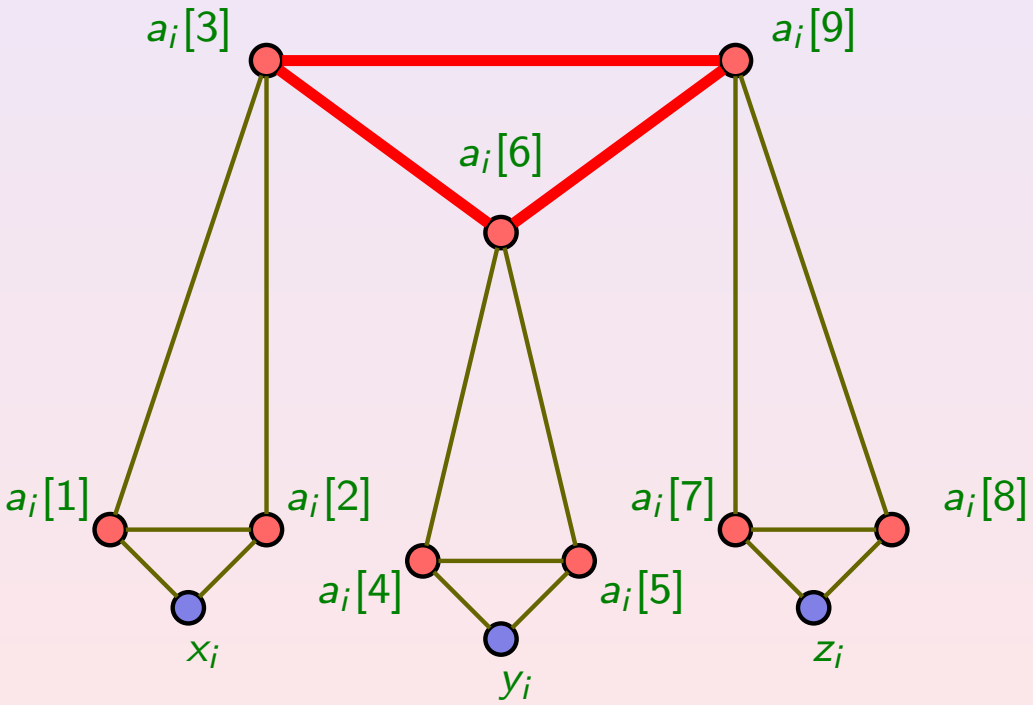
Equivalencia de los Problemas

La equivalencia de las soluciones en ambos problemas se obtiene de las dos formas que existen de elegir los triángulos en cada uno de los subgrafos de G:



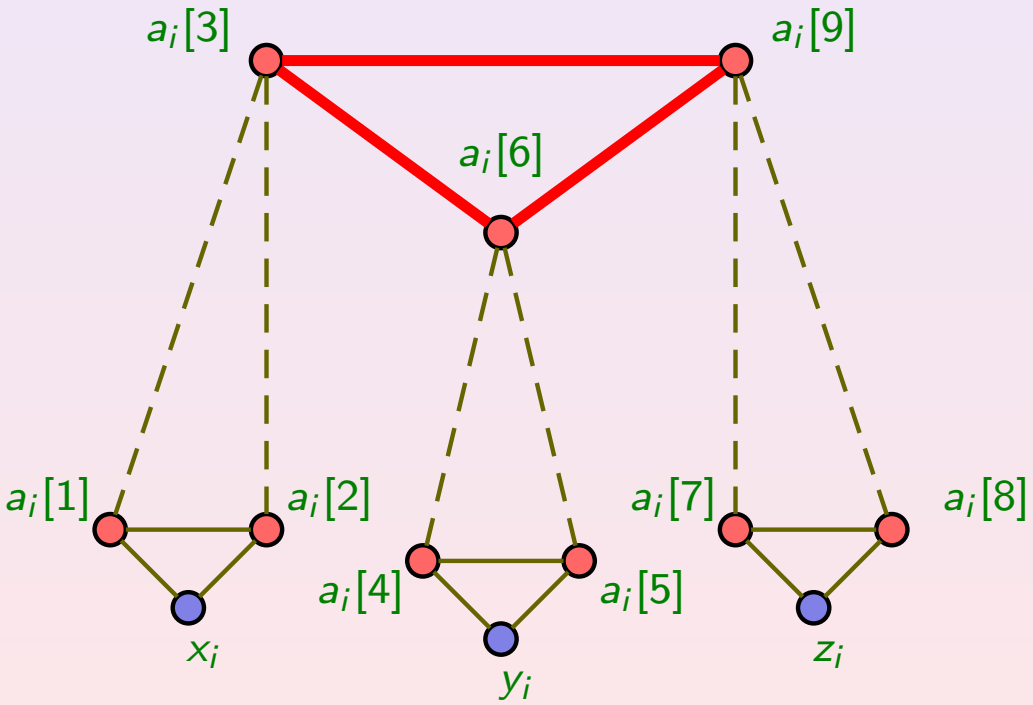
Equivalencia de los Problemas

La equivalencia de las soluciones en ambos problemas se obtiene de las dos formas que existen de elegir los triángulos en cada uno de los subgrafos de G:



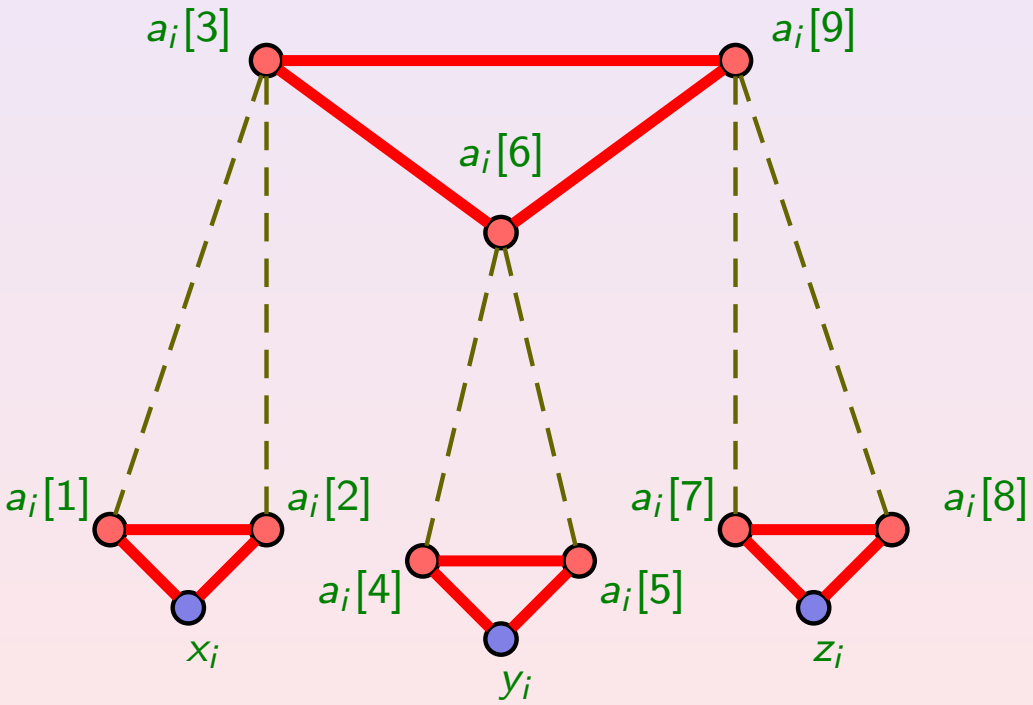
Equivalencia de los Problemas

La equivalencia de las soluciones en ambos problemas se obtiene de las dos formas que existen de elegir los triángulos en cada uno de los subgrafos de G:



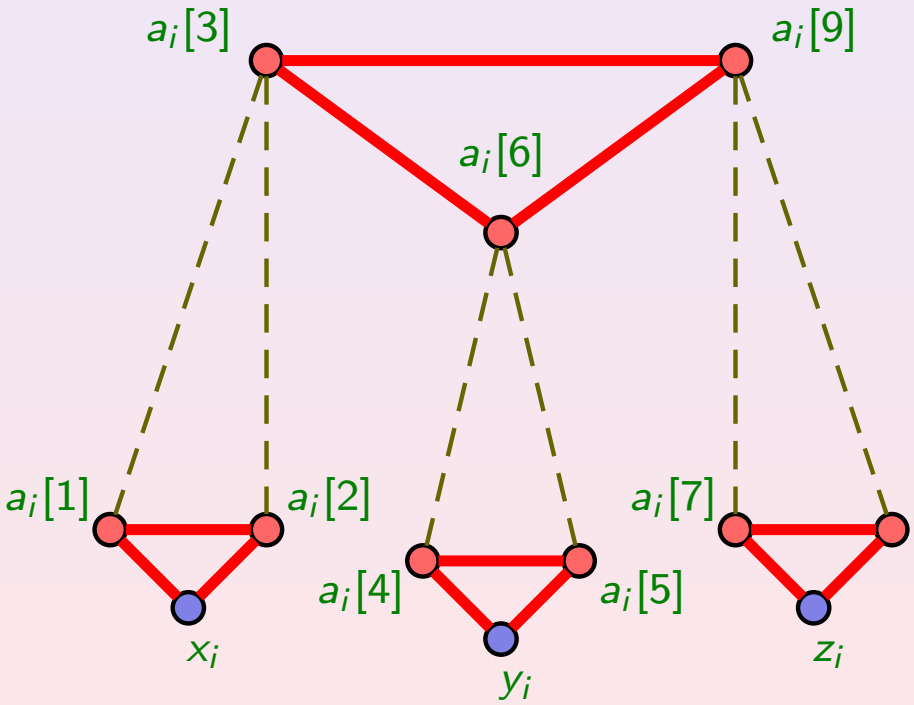
Equivalencia de los Problemas

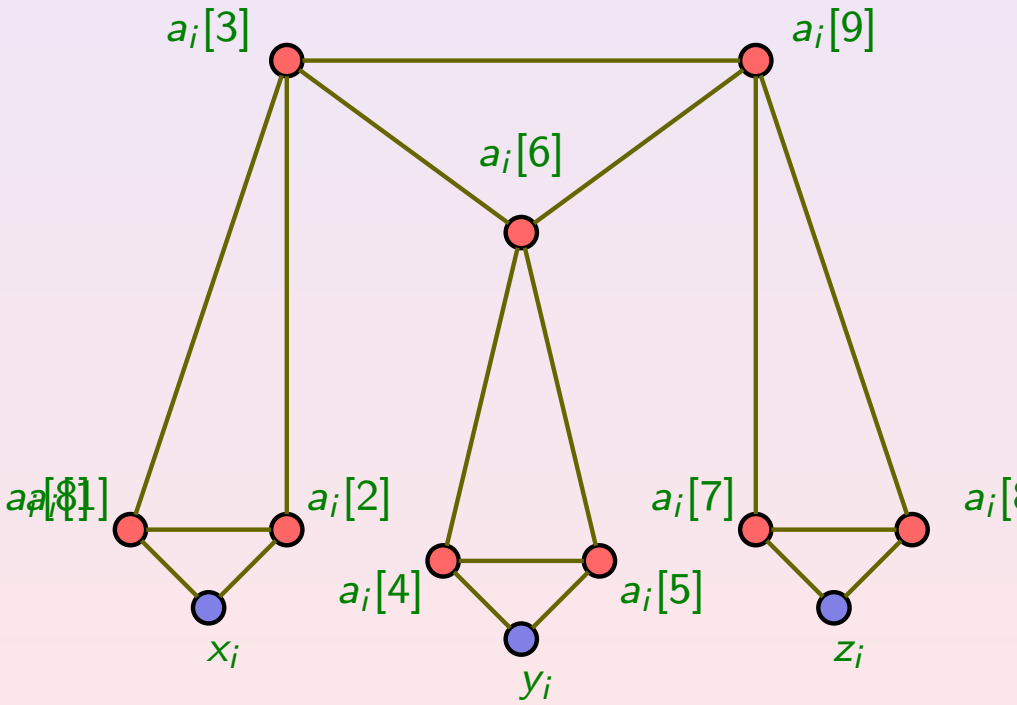
La equivalencia de las soluciones en ambos problemas se obtiene de las dos formas que existen de elegir los triángulos en cada uno de los subgrafos de G:



Equivalencia de los Problemas

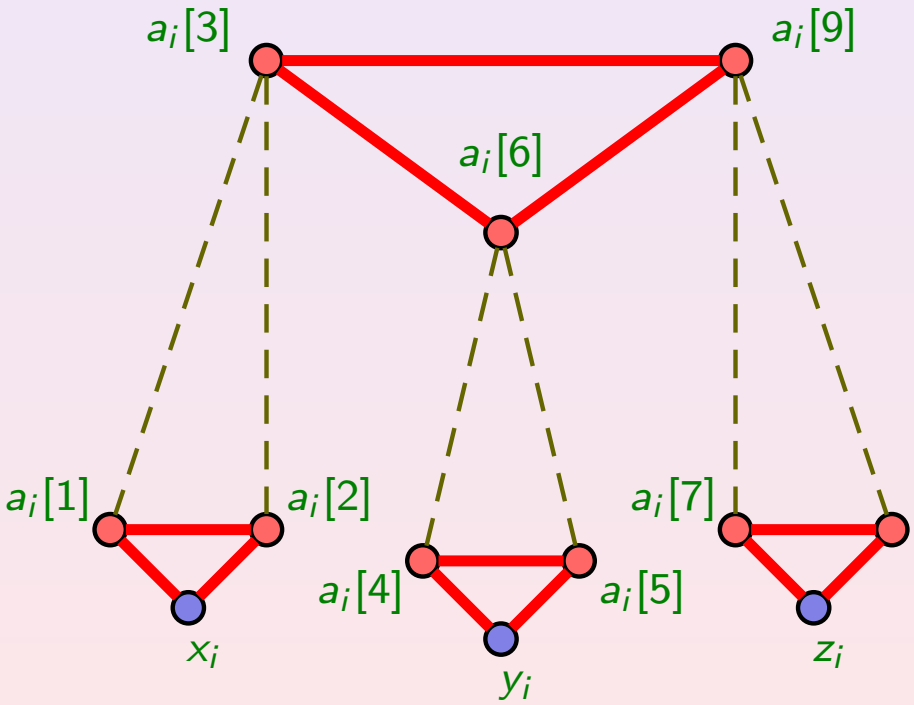
La equivalencia de las soluciones en ambos problemas se obtiene de las dos formas que existen de elegir los triángulos en cada uno de los subgrafos de G:

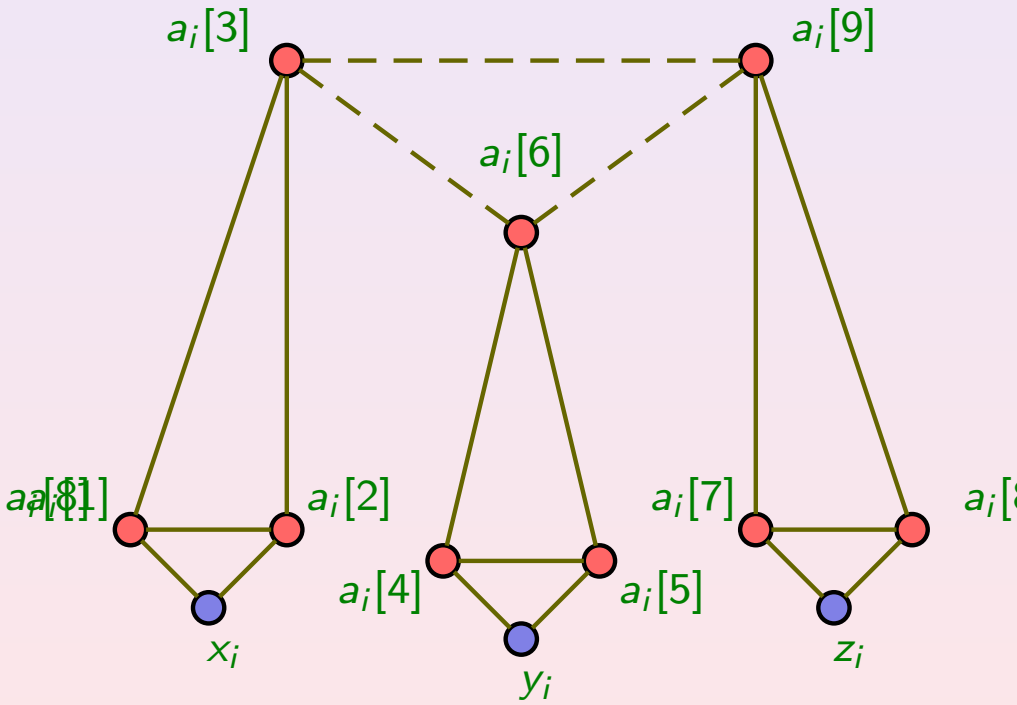




Equivalencia de los Problemas

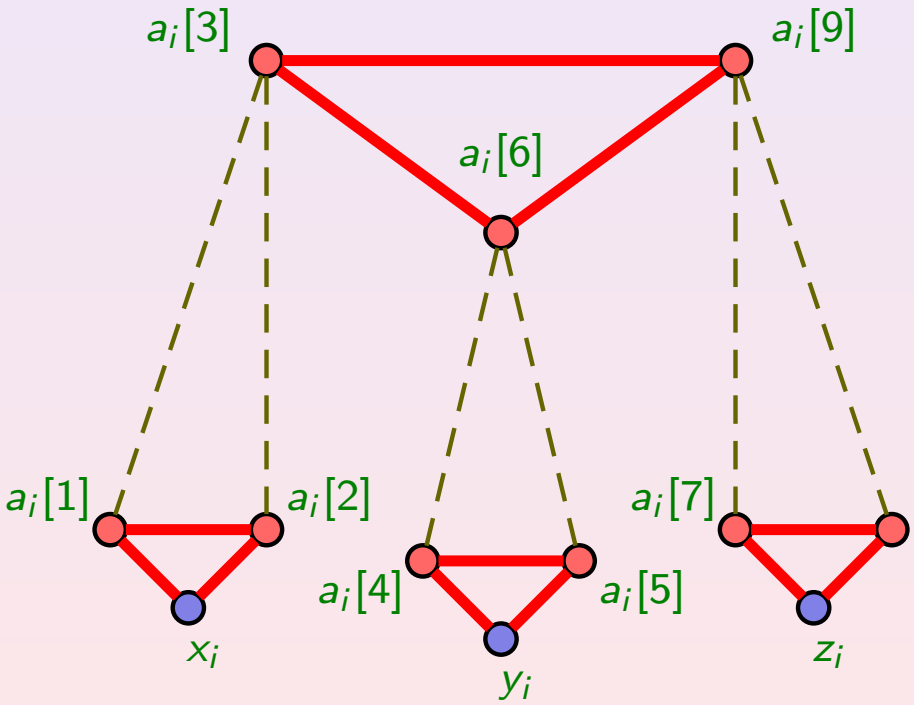
La equivalencia de las soluciones en ambos problemas se obtiene de las dos formas que existen de elegir los triángulos en cada uno de los subgrafos de G:

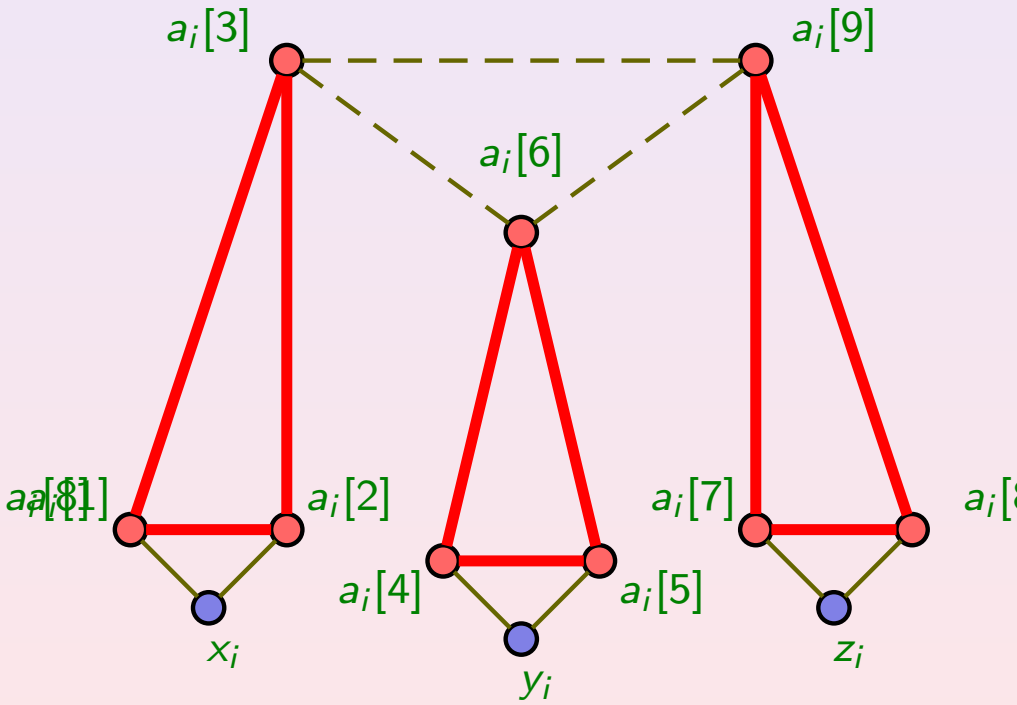




Equivalencia de los Problemas

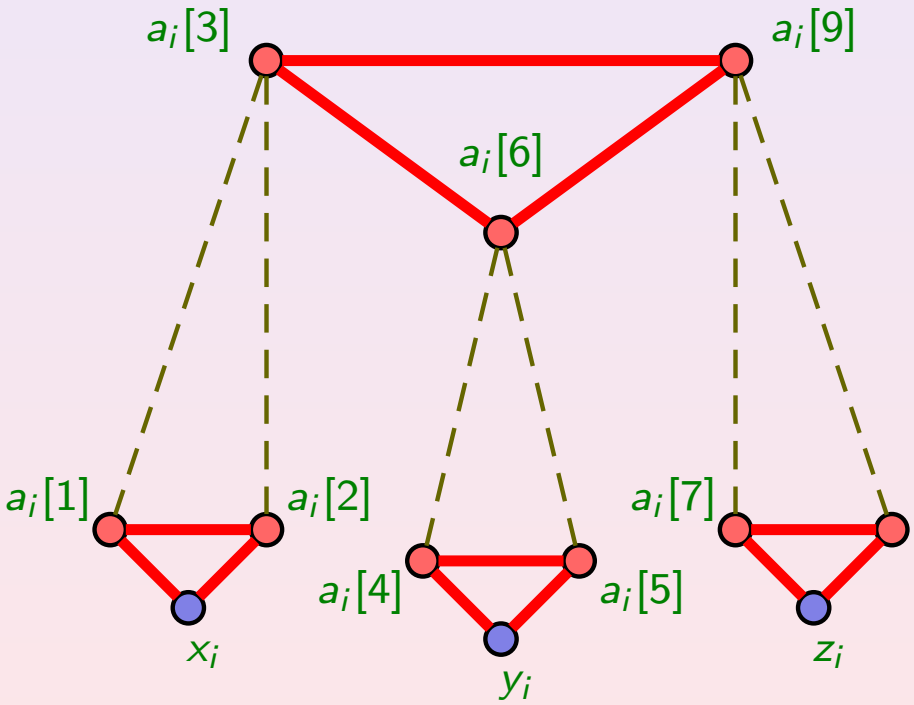
La equivalencia de las soluciones en ambos problemas se obtiene de las dos formas que existen de elegir los triángulos en cada uno de los subgrafos de G:

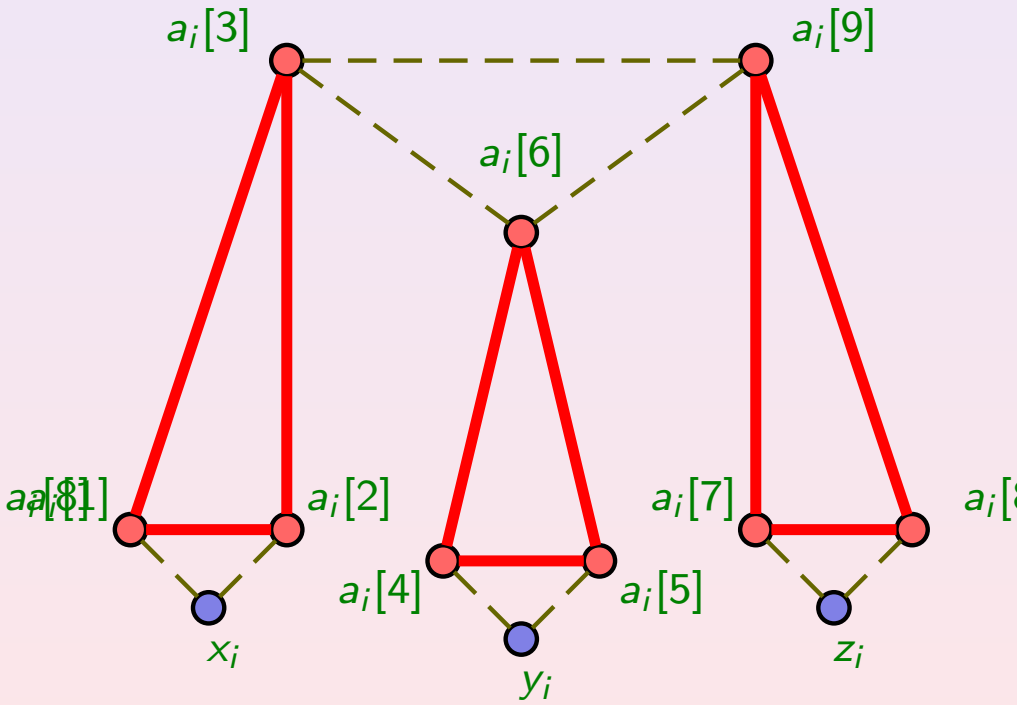




Equivalencia de los Problemas

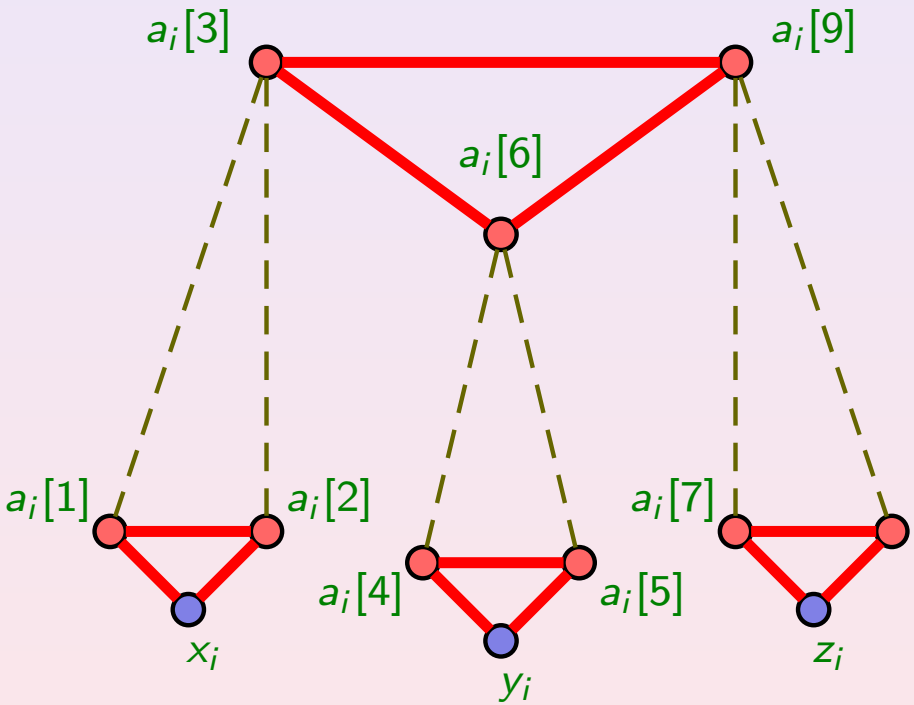
La equivalencia de las soluciones en ambos problemas se obtiene de las dos formas que existen de elegir los triángulos en cada uno de los subgrafos de G:

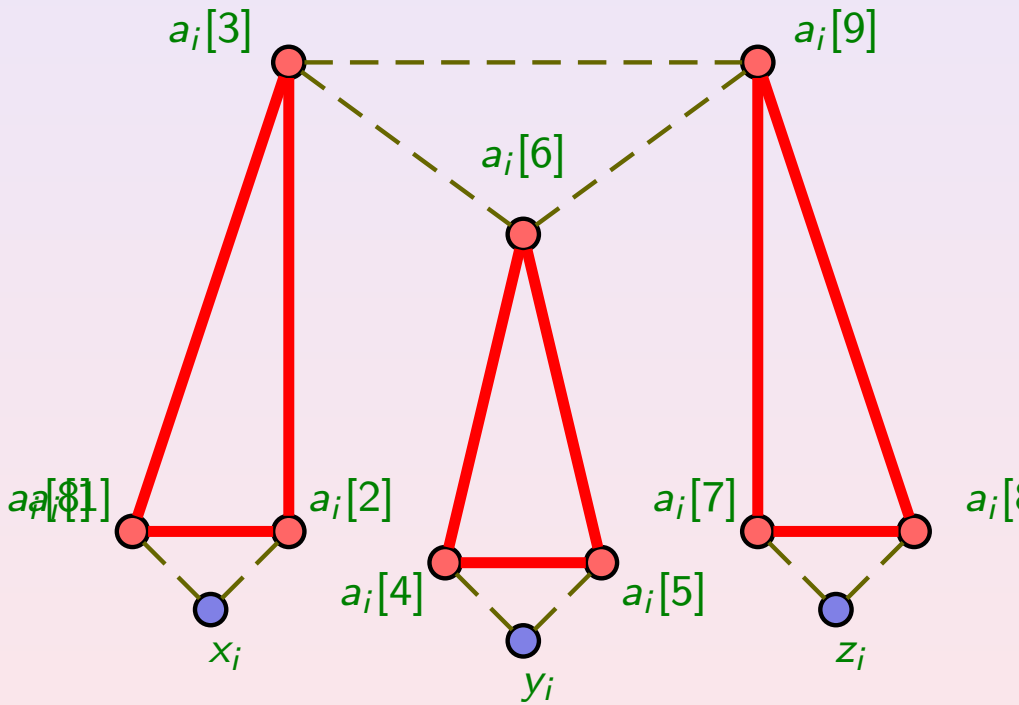




Equivalencia de los Problemas

La equivalencia de las soluciones en ambos problemas se obtiene de las dos formas que existen de elegir los triángulos en cada uno de los subgrafos de G:





La primera corresponde al caso en que $\{x_i, y_i, z_i\}$ está en el cubrimiento C', la segunda al caso en el que no está.

Reemplazamiento Local con Refuerzo

Corresponde al caso en el que, además de los elementos en que se transforman los elementos de Π', se añaden algunos elementos adicionales para forzar la equivalencia de las soluciones.

Ejemplo: La reducción del cubrimiento por tripletas al problema de la partición: ACTRI $\propto$ PARTICION.

Conjunto Mínimo de Tests (CMT)

Datos: Un conjunto finito A (posibles diagnósticos), una familia C de subconjuntos de A (posibles tests) y un entero $J \in \mathbb{N}$ que representa el número admisible de tests.
Pregunta: ¿Existe una subfamilia de test $C' \subseteq C$ con $|C'| \leq J$ y tal que para cada par de elementos distintos $a_i, a_j \in A$, existe un test $c \in C'$ que contiene uno y sólo uno de los elementos del par (un elemento, por ejemplo, a_i está en c y el otro fuera)?

Reducción: ACTRI a CMT

Vamos a reducir ACTRI a este problema. Supongamos un ejemplo de ACTRI con $M \subseteq W \times X \times Y$ y $|W| = |X| = |Y| = q$.
Vamos a crear un ejemplo de este problema equivalente a él.
Hacemos $A = W \cup X \cup Y \cup \{w_0, x_0, y_0\}$
$C = \{\{w, x, y\} : (w, x, y) \in M\} \cup \{W \cup \{w_0\}, X \cup \{x_0\}\}$
$J = q + 2$

Diseño de Componentes

Distintas componentes del problema a reducir Π' se transforman en distintas estructuras de Π que se conectan de alguna forma para forzar la equivalencia.

Ejemplos:

- Cubrimiento por vértices
- El circuito Hamiltoniano
- La demostración del teorema de Cook

Reducibilidad Turing

Un problema Π se reduce Turing a Π' lo que se representa como

$$\Pi \propto_T \Pi'$$

si y solo si Π se puede resolver en tiempo polinómico mediante un algoritmo que puede llamar a una función que resuelve Π' contando cada llamada como un paso de cálculo.

La reducibilidad Turing es un concepto *más débil* que el que hemos visto de reducibilidad espacio logarítmica: Si Π se reduce a Π', entonces se puede construir una reducibilidad Turing.

Problemas NP-Difíciles

Un problema Π es NP-difícil si y solo si existe un problema NP-completo Π' que se puede reducir (Turing) a Π:

$$\Pi' \propto_T \Pi$$

Teorema

Si un problema NP-difícil se resuelve en tiempo polinómico entonces $P = NP$.

En este tema vamos a considerar problemas NP-difíciles que tienen una dificultad *similar* a los NP-completos (existe reducción Turing entre ambos):

- Clase Co-NP: Complementarios de los problemas de NP
- Clase FNP: Problemas que buscan una solución, cuando saber si existe es NP.

La Clase Co-NP

$$CoNP = \{\overline{L} : L \in NP\}$$

Ejemplo

Dado un conjunto ϕ de fórmulas en lógica proposicional determinar si son válidas (se satisfacen para todas las asignaciones de valores de verdad).

Ejemplo

Dado un grafo determinar si NO tiene un circuito hamiltoniano.

- Estos problemas no están en NP, lo que tienen es una Máquina polinómica no determinista en la que la respuesta es afirmativa al problema si TODAS las opciones responden SI: Si la respuesta es positiva TODAS las opciones acaba en SI; si la respuesta es negativa AL MENOS UNA opción acaba en NO.
- Tampoco se reducen a los problemas NP con una transformación espacio logarítmica, ya que estas transformaciones exigen que las respuestas a ambos problemas sean **las mismas**, pero si existen reducciones Turing entre los problemas de NP y los de CoNP.

Problemas CoNP Completos

Definición

Un problema es CoNP Completo si y solo si está en la clase CoNP y cualquier otro problema de CoNP se reduce a él.

Teorema

L es NP Completo $\Leftrightarrow$ $\overline{L}$ es CoNP Completo

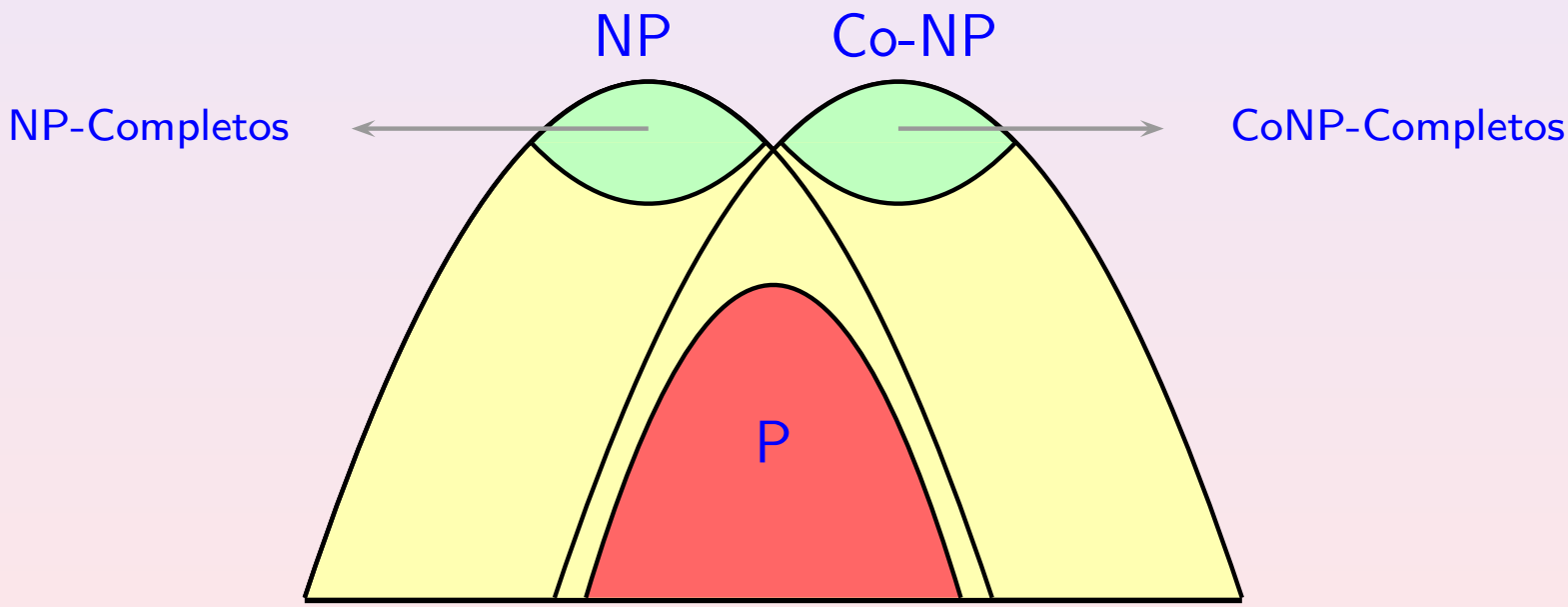
Teorema

Si un problema CoNP completo está en NP, entonces $CoNP = NP$.

Teorema

Si $P = NP$, entonces $CoNP = NP$

Estructura de Clases



Caracterizaciones

NP

Un problema $P(x)$ con entrada $x \in A^*$ está en NP si y solo si existe una relación R en $A^* \times A^*$ calculable en tiempo polinómico y un polinomio $p(n)$ tal que

$$P(x) =' Si' \Leftrightarrow \exists y \in A^* \text{ con } |y| \leq p(|x|), \quad R(x,y) = 1$$

Se dice que los problemas de NP son los problemas que se pueden *verificar* en tiempo polinómio (*de forma eficiente*). Al algoritmo que calcula R se le llama un verificador. A y se le llama un certificado.

CoNP

Un problema $P(x)$ está en CoNP si y solo si existe una relación R en $A^* \times A^*$ calculable en tiempo polinómico y un polinomio $p(n)$ tal que

$$P(x) =' Si' \Leftrightarrow \forall y \in A^* \text{ con } |y| \leq p(|x|), \quad R(x,y) = 1$$

Primalidad está en NP

Teorema

Un número $p > 1$ es primo si y solo si existe un número $1 < r < p$ tal que $r^{p-1} = 1 \bmod p$ y, además, $r^{\frac{p-1}{q}} \neq 1 \bmod p$, para todos los divisores primos q de $p-1$.

Teorema: Teorema de Pratt

Primos está en NP.

Demostración

Si p es primo, esto se puede certificar con la existencia de un número r tal que $r^{p-1} = 1 \bmod p$ y que además verificase $r^{\frac{p-1}{q}} \neq 1 \bmod p$, para todos los divisores primos q de $p-1$.

La comprobación de que $r^{p-1} = 1 \bmod p$ se puede hacer en tiempo polinómico en función de la longitud de p, que es de orden $l = \log(p)$: Para calcular 3 r^{p-1}, se calculan $r^2, r^4, \ldots, r^{2^l}$. Esto tiene un orden de $O(l^3)$.

r por sí solo no es un certificado: $20^{21-1} = 1 \bmod 21$ y 21 no es primo

Demostración

El certificado tiene que adjuntar todos los divisores primos de $p-1$. Es decir constará, en principio de r y una lista de divisores de $p-1 : (q_1, \ldots, q_k)$. El comprobar que la lista es completa se hace por divisiones sucesivas de $p-1$ entre estos números y comprobando que de uno al final.

Nos queda una cuestión, ¿cómo sabemos que los números $(q_1, \ldots, q_k)$ son primos? Pues dando certificados para ellos. Excepto para 2 que no necesita certificado.

El certificado sería una estructura recursiva: un certificado del número p sería r y una lista de divisores completa de $p-1$: $(q_1, \ldots, q_k)$, y certificados de primalidad para cada uno de ellos.

Demostración

Ejemplo

Cert(67) = ((67: 2 (2, 3, 11)), (3: 2 (2)), (11: 8 (2,5)), (5: 3 (2)))

La longitud del certificado es de orden $\log^2(p)$.

La longitud de un número y su lista es de orden $\log(p)$.

El número de listas es de orden $\log(p)$.

Problemas de Búsqueda

Sea A un alfabeto y R una relación en $A^* \times A^*$, el problema de búsqueda $P(x)$ asociado a esta relación consiste en dado un ejemplo $x \in A^*$, calcular

$$P(x) = \begin{cases} y & \text{tal que } R(x,y) \text{ si } \{z : R(x,z) = 1\} \neq \emptyset \\ \varepsilon & \text{si } \{z : R(x,z) = 1\} = \emptyset \end{cases}$$

Ejemplo

Dado un conjunto de cláusulas calcular una asignación de valores de verdad, si existe, que satisfaga todas las cláusulas.

Ejemplo

Dado un grafo calcular un circuito hamiltoniano (si este existe)

La clase FNP

NP

Un problema $P(x)$ con entrada $x \in A^*$ está en NP si y solo si existe una relación R en $A^* \times A^*$ calculable en tiempo polinómico y un polinomio $p(n)$ tal que

$$P(x) =' Si' \Leftrightarrow \exists y \in A^* \text{ con } |y| \leq p(|x|), \quad R(x,y) = 1$$

Caracterización de **FNP**

Un problema $P(x)$ está en FNP si y solo si está asociado a una relación R decidible en tiempo polinómico y tal que si $R(x,y) = 1$, entonces $|y| \leq p(|x|)$ para un polinomio p:

$$P(x) = \begin{cases} y & \text{tal que } R(x,y) \text{ si } \{z : R(x,z) = 1\} \neq \emptyset \\ \varepsilon & \text{si } \{z : R(x,z) = 1\} = \emptyset \end{cases}$$

La clase FP

FP

FP: La clase de problemas de funciones de *FNP* tales que existe una máquina de Turing determinista que las calcula en tiempo polinómico.

FNPT

FNPT: La clase de los problema de FNP totales.
Si para todo x existe un y con $|y| \leq p(|x|)$ tal que $R(x, y) = 1$

Ejemplo

Dado un número p enontrar una descomposición en números primos

$$p = p_1^{k_1}.p_2^{k_2}.....p_m^{k_m}$$

con un certificado de primalidad para cada número primo.
Seguro que existe, pero no es fácil encontrarlo.

Reducciones en **FNP**

Reducciones en **FNP**

Un problema de funciones Π se *reduce a un problema* Π' *si y solo si, existen funciones* R *y* S *calculables en espacio logarítmico, tal que para toda cadena* x *y* z *ocurre lo siguiente: Si* x *es un ejemplo de* Π *entonces* $R(x)$ *es un ejemplo de* Π'*. Además si* z *es una solución correcta de* $R(x)$ *entonces* $S(z)$ *es una solución correcta de* x*.*

Problemas **FNP**-completos

Con esto podemos definir el concepto de completitud. Un problema es **FNP**-completo si y so si está en **FNP** y cualquier otro problema de esta clase se puede reducir a este problema.

Ejemplo

Un problema **FNP**-completo: FSAT

Dado un conjunto de cláusulas, encontrar, si existe, una asignación de valores de verdad para la que todas las cláusulas sean verdaderas.

Teorema

NP = P si y solo si FNP = FP

Demostración

La demostración es muy sencilla: si $FNP = FP$ entonces es inmediato que $NP = P$.

Para la implicación inversa, supongamos que $NP = P$, entonces la consistencia se puede sesolver en tiempo polinómico.

Ahora, vamos a encontrar una asignación (si existe) en tiempo polinómico. Supongamos que tenemos un conjunto de cláusulas C. Elegimos una variable x_n, y consideramos C_1 que es C suponiendo x_n verdadero (todas las cláusulas con x_n se eliminan y en aquellas en las que aparezca $\neg x_n$ se elimina este literal) y C_2 que es C suponiendo x_n falso.

Demostración

Demostración

- Si C_1 y C_2 son inconsistentes, C también lo es y no existe la asignación de valores de verdad.
- Si C_1 es consistente, entonces si existe la asignación. Hacemos x_n verdadero y calculamos el valor de verdad de las otras variables, repitiendo de forma recursiva estos pasos sobre C_1.
- Si C_2 es consistente, entonces si existe la asignación. Hacemos x_n falso y calculamos el valor de verdad de las otras variables, repitiendo de forma recursiva estos pasos sobre C_2.

El Viajante de Comercio

Si podiésemos resolver el problema del viajante en versión de decisión en tiempo polinómico, podríamos resolver el problema de encontrar el circuito óptimo en tiempo polinómico.

- Primero se obtendría una cota superior R para el coste del circuito óptimo: el valor de distancia más grande, multiplicado por el número de ciudades.
- Después realizando una búsqueda binaria en el intervalo $[0, R]$ mediante sucesivas llamadas al problema de decisión se calcula el valor del circuito óptimo: K.

El Viajante de Comercio (Cont.)

A continuación para cada par de ciudades se hace lo siguiente:
si el coste de ir de una a otra es c, se llama al problema de decisión con presupuesto K e incrementando el coste de c a $c+1$.

- Si la respuesta es positiva, no es necesario usar este arco en el óptimo. Entonces se deja el coste a $c+1$ y se continúa.
- Si la respuesta es negativa, este arco sí se usa. Dejamos el coste al mismo valor que estaba antes y continuamos.

Al final obtenemos todos los arcos de un circuito óptimo.
Si en el primer caso se volviera al coste original, en vez de dejarlo en $c+1$, puede que no se obtenga el óptimo, ya que si hay dos circuitos óptimos ninguno de los arcos de ambos circuitos son necesarios.

Problemas de Funciones Totales

Son los de la clase **FNPT**

Un problema está en esta clase si para todo x existe un y tal que $R(x, y)$.

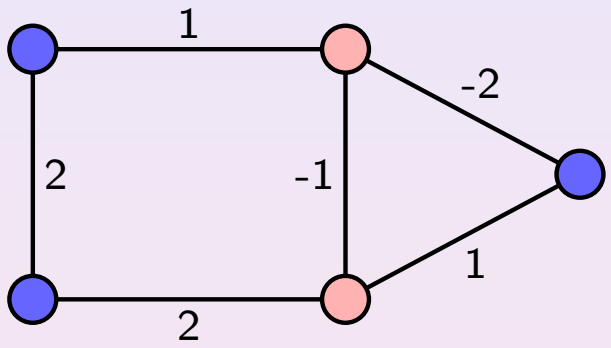
Es decir, el problema de decisión no es difícil: siempre tiene una respuesta positiva.

Eso no quiere decir que el problema sea fácil, ya que encontrar la solución no tiene por qué ser inmediato.

Muchos de ellos no se conoce que estén en FP.

La Red Feliz

Datos: Un grafo (V, E) y un peso w (que puede ser positivo o negativo para cada arco).



Un estado es una aplicación $s : V \to \{-1, 1\}$ (-1 ●, 1 ●).
El nodo i es feliz si

$$s(i).\sum_{(i,j)\in E} s(j).w(i,j) \geq 0$$

Solución: Un estado s en el que todos los nodos sean felices.

La Red Feliz

El problema está en FNP y es total.
Consideremos

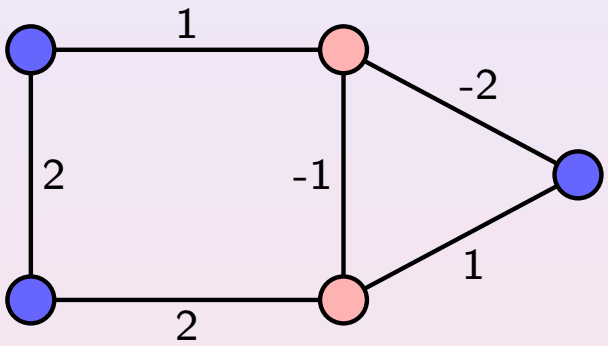
$$\Phi(s) = \sum_{(i,j)\in E} s(i)s(j)w(i,j)$$

Si tenemos un nodo infeliz, i, en s, entonces

$$s(i).\sum_{(i,j)\in E} s(j).w(i,j) = -\delta < 0$$

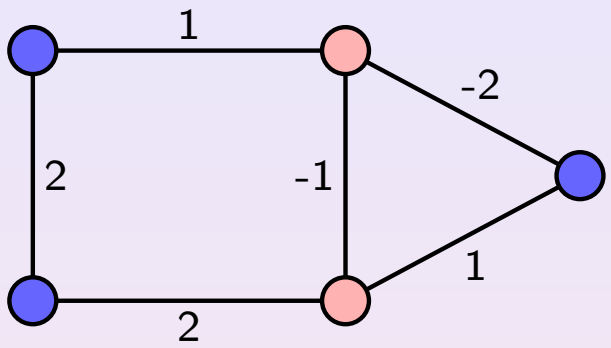
Sea el estado s' igual que s, excepto que $s'(i) = -s(i)$.
Entonces $\Phi(s') = \Phi(s) + 2\delta$.
Como el valor de Φ no puede crecer indefinidamente, este proceso tiene que terminar con una red feliz.
Este algoritmo es pseudo polinómico, pero no polinómico.

Ejemplo



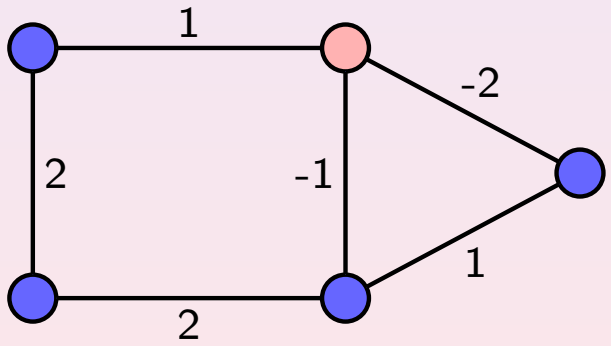
$\Phi = -1$

Ejemplo



$\Phi = -1$

Cambiamos un nodo infeliz:



$\Phi = 7$

La red ya es feliz.

Estructura de Clases de Funciones