Programación 3 - Reducciones de tiempo polinomial

Instituto de Computación

Reducciones de tiempo polinomial

- Problema Y se reduce a X en tiempo polinomial si se puede resolver mediante un algoritmo que:
  - Realiza una cantidad polinomial (en el tamaño de la instancia de Y) de invocaciones a un algoritmo que resuelve X.
  - El tiempo de cómputo adicional, excluyendo estas llamadas, es polinomial en el tamaño de la instancia de Y.
- En este caso escribimos $Y \leq_P X$. Consecuencias:
  - Si X se puede resolver en tiempo polinomial, entonces Y también.
  - Si Y NO se puede resolver en tiempo polinomial, entonces X tampoco.
- Reducciones a la Karp: se realiza una única llamada al algoritmo que resuelve X y la salida se usa directamente como respuesta al problema original.

Reducción a la Karp de problemas de decisión

- Un problema de decisión consiste en responder una pregunta binaria (con respuesta sí / no) a partir de ciertas entradas.
- Ejemplo: Dado un grafo G, ¿es G bipartito?
- Para problemas de decisión X, Y, probar que $Y \leq_P X$ mediante reducción a la Karp de Y a X consiste en:
  1. Definir un algoritmo A que, dada una instancia y del problema Y, construye una instancia $x = A(y)$ del problema X.
     En otras palabras, A transforma las entradas para un algoritmo que resuelve Y en entradas para un algoritmo que resuelve X.
  2. Mostrar que A termina en tiempo polinomial en el tamaño de la entrada y.
  3. Mostrar que, para toda instancia y de Y, se cumple que la respuesta del problema Y para la instancia y es igual a la respuesta del problema X para la instancia $A(y)$.
- **Observación**: No nos interesan las instancias x que no son imagen de ningún y.

¿Para qué usamos reducciones?

Supongamos que tenemos entre manos un problema X.

¿Para qué usamos reducciones?

Supongamos que tenemos entre manos un problema X.

- Para encontrar un algoritmo que resuelve X:
  - Buscamos un problema Y que sabemos cómo resolver y
  - reducimos X a Y.

¿Para qué usamos reducciones?

Supongamos que tenemos entre manos un problema X.

- Para encontrar un algoritmo que resuelve X:
  - Buscamos un problema Y que sabemos cómo resolver y
  - reducimos X a Y.
  - Esto es lo que venimos haciendo hasta ahora.

¿Para qué usamos reducciones?

Supongamos que tenemos entre manos un problema X.

- Para encontrar un algoritmo que resuelve X:
  - Buscamos un problema Y que sabemos cómo resolver y
  - reducimos X a Y.
  - Esto es lo que venimos haciendo hasta ahora.

- Para probar que X es "difícil" (no se conoce algoritmo eficiente para resolverlo):
  - Buscamos un problema Y que sabemos que es difícil
  - reducimos Y a X para probar que $Y \leq_P X$.

¿Para qué usamos reducciones?

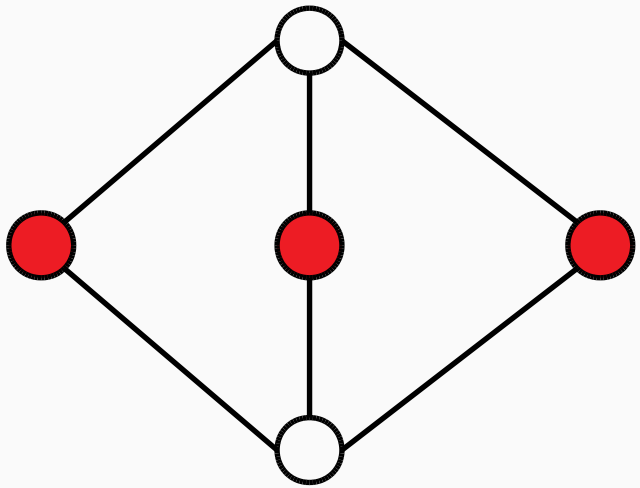
Supongamos que tenemos entre manos un problema X.

- Para encontrar un algoritmo que resuelve X:
  - Buscamos un problema Y que sabemos cómo resolver y
  - reducimos X a Y.
  - Esto es lo que venimos haciendo hasta ahora.

- Para probar que X es "difícil" (no se conoce algoritmo eficiente para resolverlo):
  - Buscamos un problema Y que sabemos que es difícil
  - reducimos Y a X para probar que $Y \leq_P X$.
  - Esto es lo que vamos a hacer ahora!

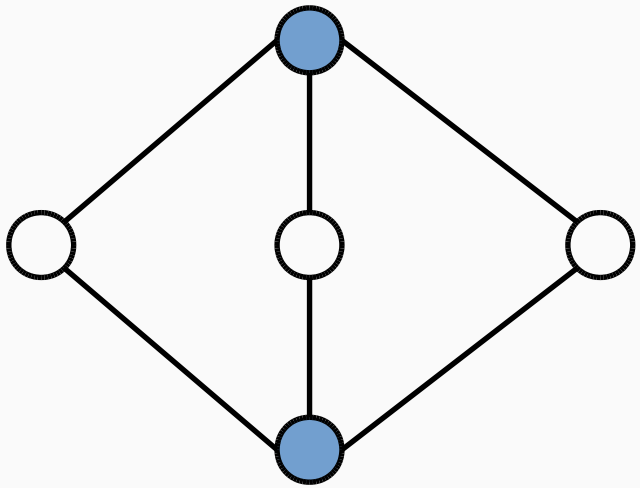
Independent Set (Conjunto Independiente)

- Sea $G = (V, E)$ un grafo.
- $S \subset V$ es un *IS* sii para todo $u \in S$ los vértices adyacentes a u no pertenecen a S.
- $S = \emptyset$ es trivialmente un *IS*, en general nos interesa un *IS* lo más grande posible.
- **Problema de decisión:** Dado G y un natural k, ¿existe un *IS* de tamaño al menos k?



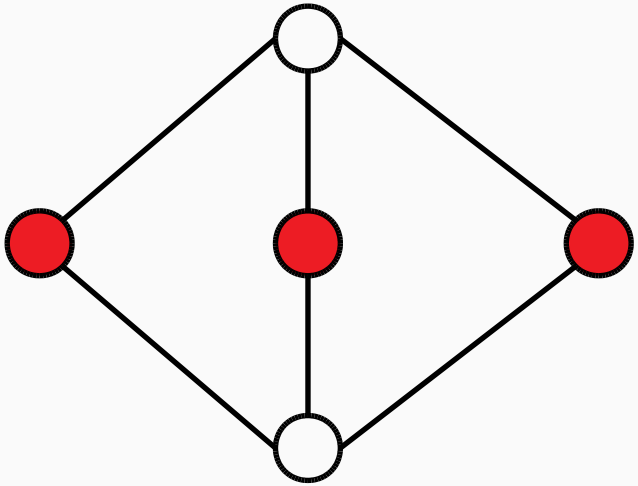
Vertex Cover (Cobertura de aristas por vértices)

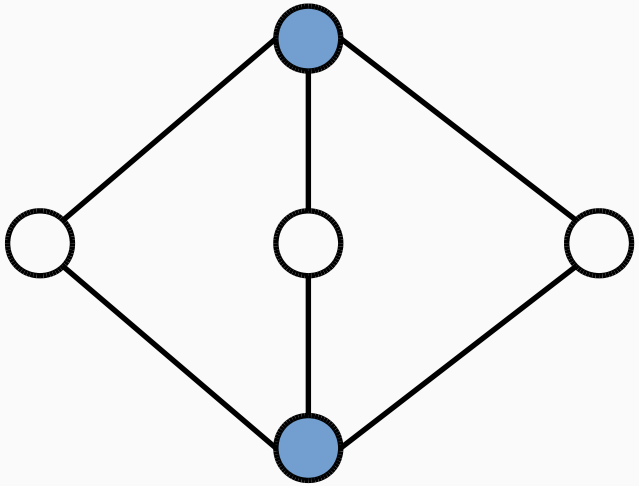
- Sea $G = (V, E)$ un grafo.
- $S \subset V$ es un VC sii toda arista tiene al menos uno de sus vértices en S.
- $S = V$ es trivialmente un VC, en general nos interesa un VC lo más chico posible.
- **Problema de decisión:** Dado G y un natural k, ¿existe un VC de tamaño no mayor a k?



Independent Set / Vertex Cover

- $S \subset V$ es un *IS* sii para todo $u \in S$ los vértices adyacentes a u no pertenecen a S.
- Equivalentemente, S es un *IS* sii en toda arista se cumple que alguno de sus vértices no pertenece a S.
- Es decir, sii $V \setminus S$ es un *VC*.



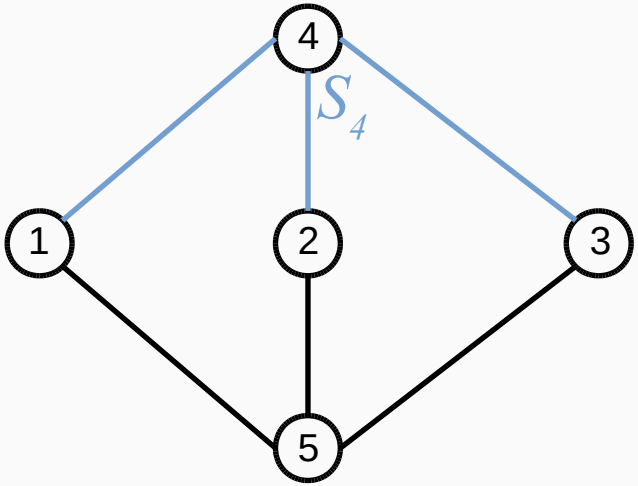


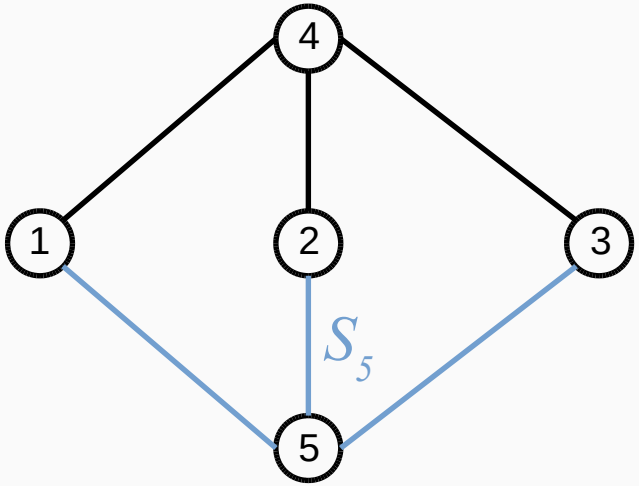
Reducción mutua entre Independent Set y Vertex Cover

- Como S es un IS sii $V \setminus S$ es un VC, entonces
- existe un IS de tamaño al menos k sii existe un VC de tamaño no mayor a $n - k$.
- Podemos resolver *Independent Set*(G, k) resolviendo *Vertex Cover*$(G, n - k)$ y viceversa.
- La reducción de tiempo polinomial (a la Karp) de uno a otro es trivial: cambiamos k en una instancia por $n - k$ en la otra.
- En consecuencia se cumple *Independent Set* $\leq_P$ *Vertex Cover* y también *Vertex Cover* $\leq_P$ *Independent Set*.

Generalizaciones

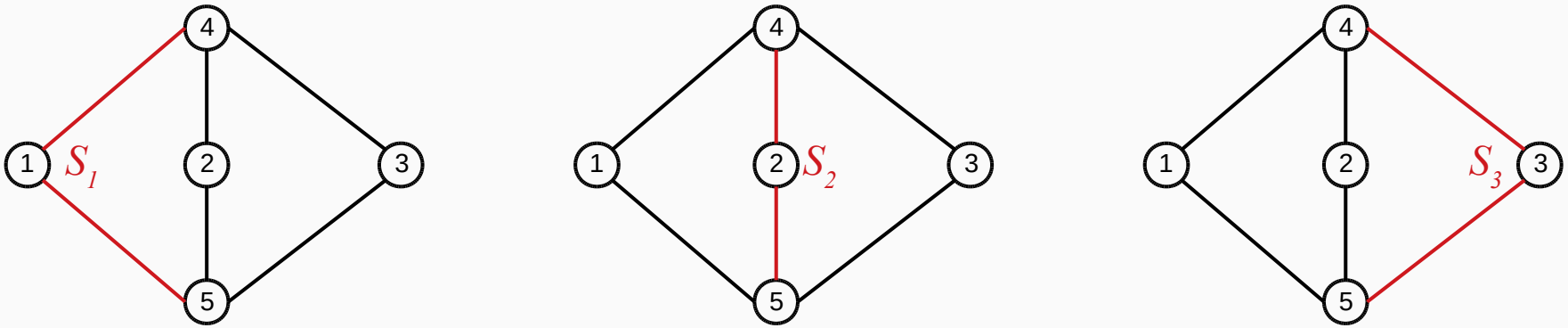
- **Cobertura de conjuntos (Set Cover):** Dado un conjunto finito U, un natural r, y dados $S_1, S_2, , \ldots S_\ell$, subconjuntos de U, ¿es posible elegir no más de r de estos subconjuntos de modo que la unión de todos ellos sea igual a U?
- *Vertex Cover* $\leq_P$ *Set Cover*: Dado G y k, reducimos *Vertex Cover* a *Set Cover* tomando
  - $U = E$,
  - $r = k$,
  - $S_i = \{(u, v) \in E : u = i\}$, para todo $i \in V$.





Generalizaciones

- **Empaquetado de conjuntos (Set Packing):** Dado un conjunto finito U, un natural r, y dados $S_1, S_2, , \ldots S_\ell$, subconjuntos de U, ¿es posible elegir al menos r de estos subconjuntos que sean disjuntos entre sí?
- *Independent Set* $\leq_P$ *Set Packing*: Dado G y k, reducimos *Independent Set* a *Set Packing* tomando
  - $U = E$,
  - $r = k$,
  - $S_i = \{(u, v) \in E : u = i\}$, para todo $i \in V$.



Problemas de decisión, búsqueda y optimización

- Decisión: ¿**Existe** VC de tamaño no superior a k?
- Búsqueda: **Encontrar** un VC de tamaño no superior a k.
- Optimización: Encontrar un VC de tamaño **mínimo**.

Decisión en tiempo polinomial $\implies$ búsqueda en tiempo polinomial:

1. Si G no tiene aristas, devolver $\emptyset$.
2. Si no existe VC de tamaño k, responder *no existe* y terminar.
3. Buscar $v \in V$ t.q. $G - \{v\}$ tiene VC de tamaño $k-1$.
4. Obtener (recursivamente) S, VC de $G - \{v\}$ de tamaño $k-1$.
5. Devolver $S \cup \{v\}$.

Búsqueda en tiempo polinomial $\implies$ Optimización en tiempo polinomial:

- Bipartición en k.

SAT

- $X = \{x_1, \ldots, x_n\}$ conjunto de variables booleanas (0/1).
- Una *cláusula* es una disyunción de términos

$$C_i = t_{i,1} \vee t_{i,2} \vee \ldots \vee t_{i,r},$$

  donde cada *término* $t_{i,j}$ es una variable x o su negación $\bar{x}$.
- Una *asignación de verdad* para X es una asignación de valores en $\{0, 1\}$ para cada una de las variables de X.
- Una asignación satisface la cláusula C_i si hace que al evaluar C_i con esos valores asignados a las variables de X la cláusula valga 1.
- Por ejemplo, para la cláusula $x_1 \vee \bar{x_2}$, la única asignación de verdad que no la satisface es $x_1 = 0$, $x_2 = 1$.

SAT

- **Problema de decisión (SAT):** Dado un conjunto de cláusulas $C_1, \ldots, C_k$ sobre un conjunto de variables booleanas X, ¿existe una asignación de verdad para X que satisfaga simultáneamente **todas** las cláusulas?
- Equivalentemente, ¿existe una asignación de verdad para X que satisfaga la conjunción $\Phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$?
- Ejemplo: No es posible satisfacer

$$\Phi = (x_1 \vee x_2) \wedge (x_1 \vee \bar{x_2}) \wedge (\bar{x_1} \vee x_2) \wedge (\bar{x_1} \vee \bar{x_2})$$

3SAT

- **Problema de decisión (3SAT):** Es una restricción de *SAT* en la cual todas las cláusulas tienen **exactamente 3 términos** sobre **3 variables distintas**.
- Claramente se cumple $3SAT \leq_P SAT$ porque las instancias de *3SAT* son casos particulares de *SAT*.
- Para mostrar que también se cumple $SAT \leq_P 3SAT$:
  1. Definimos un algoritmo A que:
     - Dado un conjunto de cláusulas de *SAT*, $\mathcal{C} = C_1, \ldots, C_k$ sobre un conjunto de variables X,
     - construye un conjunto de cláusulas de *3SAT*, $\mathcal{C}' = C'_1, \ldots, C'_{k'}$ sobre un conjunto de variables X'.
  2. Probamos que, para todo $\mathcal{C}, X$, existe una asignación de verdad para X que satisface $\mathcal{C}$ si y solo si existe una asignación de verdad para X' que satisface $\mathcal{C}'$, donde $\mathcal{C}', X'$ resultan de aplicar A a $\mathcal{C}, X$.
  3. Probamos que A corre en tiempo polinomial.

SAT $\leq_P$ 3SAT

Dada una instancia de *SAT*:

- Eliminamos de $\mathcal{C}$ todas las cláusulas que contengan simultáneamente una variable y su negación. Por ejemplo $C_i = x_1 \vee \bar{x_1}$.
- Eliminamos todos los términos repetidos de cada cláusula de $\mathcal{C}$ dejando solo una copia de cada término en cada cláusula. Por ejemplo $C_i = x_1 \vee x_2 \vee x_2$ se sustituye por $C_i = x_1 \vee x_2$.
- La instancia de *SAT* obtenida con estas modificaciones no tiene tiene ninguna variable repetida en ninguna cláusula, y es equivalente a la original, en el sentido de que puede ser satisfecha si y solo si la instancia original puede ser satisfecha.

SAT $\leq_P$ 3SAT

A partir de esta instancia de *SAT* construimos una de *3SAT* de la siguiente manera:

- Comenzamos con $X' = X$ y $\mathcal{C}'$ vacío.
- Agregamos variables, u, w, y las cláusulas

$$C'_{0,1} = \overline{u} \vee w \vee x_1 \qquad C'_{0,5} = \overline{w} \vee u \vee x_1$$
$$C'_{0,2} = \overline{u} \vee w \vee \bar{x_1} \qquad C'_{0,6} = \overline{w} \vee u \vee \bar{x_1}$$
$$C'_{0,3} = \overline{u} \vee \bar{w} \vee x_1$$
$$C'_{0,4} = \overline{u} \vee \bar{w} \vee \bar{x_1}$$

- Para cada cláusula de $\mathcal{C}$ formada por un único término, $C_i = t_{i,1}$, agregamos a $\mathcal{C}'$ la cláusula $t_{i,1} \vee u \vee w$.
- Para cada cláusula de $\mathcal{C}$ formada por dos términos, $C_i = t_{i,1} \vee t_{i,2}$, agregamos a $\mathcal{C}'$ la cláusula $t_{i,1} \vee t_{i,2} \vee u$.

SAT $\leq_P$ 3SAT

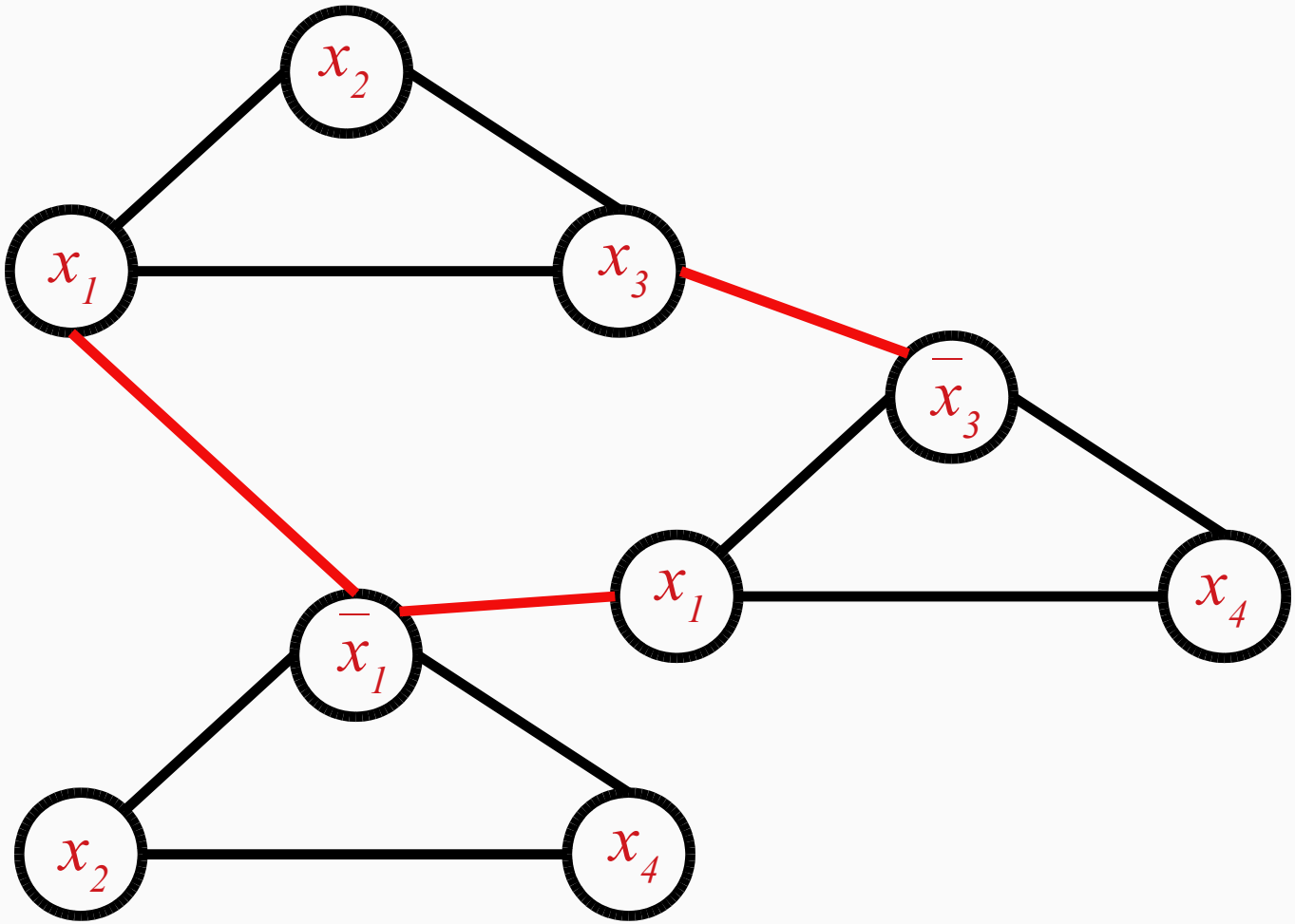
- Para cada cláusula de $\mathcal{C}$ con 3 términos agregamos una igual en $\mathcal{C}'$.
- Para cada cláusula de $\mathcal{C}$ con más de 3 términos, $C_i = t_{i,1} \vee t_{i,2} \vee \ldots \vee t_{i,r}$, agregamos a X' las variables $z_{i,2}, z_{i,3}, \ldots, z_{i,r-2}$ y agregamos a $\mathcal{C}'$ las cláusulas

$$
\begin{aligned}
C'_{i,2} &= t_{i,1} \vee t_{i,2} \vee z_{i,2} \\
C'_{i,3} &= \overline{z_{i,2}} \vee t_{i,3} \vee z_{i,3} \\
C'_{i,4} &= \overline{z_{i,3}} \vee t_{i,4} \vee z_{i,4} \\
C'_{i,5} &= \overline{z_{i,4}} \vee t_{i,5} \vee z_{i,5} \\
&\ldots \\
C'_{i,r-2} &= \overline{z_{i,r-3}} \vee t_{i,r-2} \vee z_{i,r-2} \\
C'_{i,r-1} &= \overline{z_{i,r-2}} \vee t_{i,r-1} \vee t_r
\end{aligned}
$$

3SAT $\leq_P$ Independent Set

Reducción de *3SAT* a *Independent Set* para

$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x_3} \vee x_4) \wedge (\bar{x_1} \vee x_2 \vee x_4)$$



Resumen

- Algunos problemas tipo:
  - Empaquetado (*Independent Set*, *Set Packing*).
  - Cobertura (*Vertex Cover*, *Set Cover*).
  - Satisfacción de restricciones (*SAT*, *3SAT*).

Resumen

- Algunos problemas tipo:
  - Empaquetado (*Independent Set*, *Set Packing*).
  - Cobertura (*Vertex Cover*, *Set Cover*).
  - Satisfacción de restricciones (*SAT*, *3SAT*).
- Algunas estrategias:
  - Equivalencia sencilla. Ej. *Independent Set* $=_P$ *Vertex Cover*.
  - Caso particular. Ej. *Vertex Cover* $\leq_P$ *Set Cover*.
  - Reducción mediante artilugios. Ej. *3SAT* $\leq_P$ *Independent Set*.
  - Transitividad:
    3SAT $\leq_P$ *Independent Set* $\leq_P$ *Vertex Cover* $\leq_P$ *Set Cover*.

Resumen

- Algunos problemas tipo:
  - Empaquetado (*Independent Set*, *Set Packing*).
  - Cobertura (*Vertex Cover*, *Set Cover*).
  - Satisfacción de restricciones (*SAT*, *3SAT*).
- Algunas estrategias:
  - Equivalencia sencilla. Ej. *Independent Set* $=_P$ *Vertex Cover*.
  - Caso particular. Ej. *Vertex Cover* $\leq_P$ *Set Cover*.
  - Reducción mediante artilugios. Ej. *3SAT* $\leq_P$ *Independent Set*.
  - Transitividad:
    3SAT $\leq_P$ *Independent Set* $\leq_P$ *Vertex Cover* $\leq_P$ *Set Cover*.
- ¿Hacia dónde vamos?
  - Todos estos problemas son de dificultad equivalente.
  - Están en la "cúspide" de dificultad de una amplia familia de problemas, que incluye a todos los que se resuelven en tiempo polinomial.
  - No sabemos si existe algún algoritmo de tiempo polinomial para resolverlos.