Máquinas de Turing no-deterministas (MTND)

- Una MTND tiene los mismos componentes que vimos para una MTD, con la siguiente excepción.
- Un programa en una MTND es una tabla que mapea un par (q_i, t_i) a un conjunto de ternas $(q_f, t_f, \{0, +1, -1\})$.
- Una MTND resuelve un problema de decisión si existe una secuencia de alternativas que lleva a un estado de aceptación si y sólo si la respuesta es "sí".

Clases P y NP

- Un problema de decisión está en la **clase P** si las instancias "sí" son reconocidas por una MTD polinomial.
- Un problema de decisión está en la **clase NP** si las instancias "sí" son reconocidas por una MTND polinomial.
- Alternativamente, La clase NP se puede definir como el conjunto de problemas de decisión que admiten un certificado polinomial.
- Relaciones entre las clases:
  1. $P \subseteq NP$
  2. **Problema abierto:** ¿Es $P = NP$?

Ejemplo: Conjunto independiente máximo

Dado un grafo $G = (V, X)$ y un entero k, ¿tiene G un conjunto independiente de tamaño mayor o igual a k?

guess(S): función multivaluada que retorna un nuevo elemento de S.

$I := \emptyset$
$S := V$
mientras $S \neq \emptyset$ **hacer**
    $v :=$ *guess*(S)
    $S := S \setminus \{v\}$
    si $I \cup \{v\}$ `es independiente` **entonces** $I := I \cup \{v\}$
    si $|I| \geq k$ **entonces retornar** ‘‘`sí`’’
fin mientras
retornar ‘‘`no`’’

Transformaciones polinomiales

Todavía no se demostró que exista un problema en NP $\setminus$ P. Mientras tanto, se estudian clases de complejidad "relativa", es decir, que establecen orden de dificultad entre problemas.

- Una transformación o reducción polinomial de un problema de decisión Π_1 a uno Π_2 es una función que se computa en tiempo polinomial y transforma una instancia I_1 de Π_1 en una instancia I_2 de Π_2 tal que I_1 tiene respuesta "sí" para Π_1 si y sólo si I_2 tiene respuesta "sí" para Π_2.

- El problema de decisión Π_1 se reduce polinomialmente a otro problema de decisión Π_2, $\Pi_1 \leq_p \Pi_2$, si existe una transformación polinomial de Π_1 a Π_2.

Si $\Pi'' \leq_p \Pi'$ y $\Pi' \leq_p \Pi$ entonces $\Pi'' \leq_p \Pi$, ya que la composición de dos reducciones polinomiales es una reducción polinomial.

Problemas NP-completos

Definición:

Un problema Π es **NP-completo** si:

1. $\Pi \in \text{NP}$.
2. Para todo $\Pi' \in \text{NP}$, $\Pi' \leq_p \Pi$.

¿P $\neq$ NP? La pregunta del millón...

- **¿Qué pasa si existe un problema en NP-c $\cap$ P?**

¿P $\neq$ NP? La pregunta del millón...

- **¿Qué pasa si existe un problema en NP-c $\cap$ P?**
- **Implicaría que P=NP.**

¿P $\neq$ NP? La pregunta del millón...

- **¿Qué pasa si existe un problema en NP-c $\cap$ P?**
- **Implicaría que P=NP.**
  - Si $\Pi \in$ NP-c $\cap$ P, existe un algoritmo polinomial que resuelve Π, por estar Π en P. Por otro lado, como Π es NP-completo, para todo $\Pi' \in$ NP, $\Pi' \leq_p \Pi$.

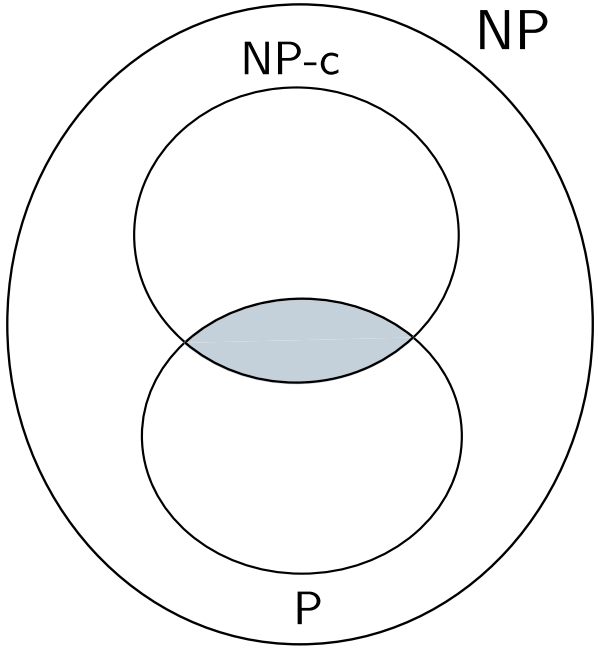
¿P $\neq$ NP? La pregunta del millón...

- **¿Qué pasa si existe un problema en NP-c $\cap$ P?**
- **Implicaría que P=NP.**
  - Si $\Pi \in$ NP-c $\cap$ P, existe un algoritmo polinomial que resuelve Π, por estar Π en P. Por otro lado, como Π es NP-completo, para todo $\Pi' \in$ NP, $\Pi' \leq_p \Pi$.
  - Sea $\Pi' \in$ NP. Apliquemos la reducción polinomial que transforma instancias de Π' en instancias de Π y luego el algoritmo polinomial que resuelve Π. Por definición de reducción polinomial, es fácil ver que lo que se obtiene es un algoritmo polinomial que resuelve Π'.

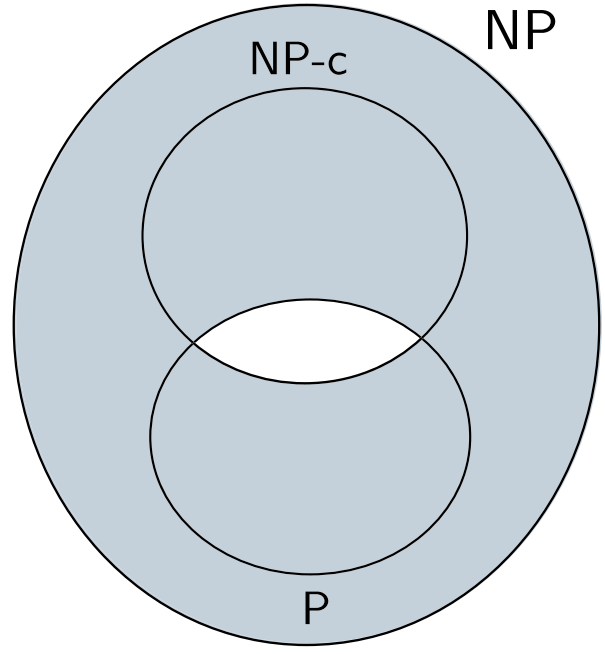
¿P $\neq$ NP? La pregunta del millón...

- **¿Qué pasa si existe un problema en NP-c $\cap$ P?**
- **Implicaría que P=NP.**
  - Si $\Pi \in$ NP-c $\cap$ P, existe un algoritmo polinomial que resuelve Π, por estar Π en P. Por otro lado, como Π es NP-completo, para todo $\Pi' \in$ NP, $\Pi' \leq_p \Pi$.
  - Sea $\Pi' \in$ NP. Apliquemos la reducción polinomial que transforma instancias de Π' en instancias de Π y luego el algoritmo polinomial que resuelve Π. Por definición de reducción polinomial, es fácil ver que lo que se obtiene es un algoritmo polinomial que resuelve Π'.
- Hasta el momento no se conoce ningún problema en NP-c $\cap$ P, así como tampoco se ha demostrado que un problema esté en NP $\setminus$ P. En ese caso, obviamente, se probaría que P $\neq$ NP.

Esquema de clases



si P=NP...



si P≠NP...

¿Cómo se prueba que un problema es NP-completo?

El problema SAT consiste en decidir si, dada una fórmula lógica φ expresada como conjunción de disyunciones (ej: $\varphi = x_1 \wedge (x_2 \vee \neg x_1) \wedge (x_3 \vee \neg x_4 \vee x_1)$), existe una valuación de sus variables que haga verdadera φ.

Teorema de Cook-Levin (1971): SAT es NP-completo.

La demostración de Cook es directa: considera un problema genérico $\Pi \in$ NP y una instancia genérica $d \in D_\Pi$. A partir de la hipotética NDTM que resuelve Π, genera en tiempo polinomial una fórmula lógica $\varphi_{\Pi,d}$ en forma normal (conjunción de disyunciones) tal que d tiene respuesta "sí" en Π si y sólo si $\varphi_{\Pi,d}$ es satisfactible.

Teorema de Cook-Levin (idea de demostración)

La idea es escribir una fórmula que exprese exactamente que la NDTM $\mathcal{M}$ que resuelve Π, para el input d, termina en un estado q_{si}.

Teorema de Cook-Levin (idea de demostración)

La idea es escribir una fórmula que exprese exactamente que la NDTM $\mathcal{M}$ que resuelve Π, para el input d, termina en un estado q_{si}.

Para eso, se definen variables que van a expresar si el i-ésimo casillero de la cinta contiene cierto valor en el j-ésimo paso o no, si la cabeza de la máquina está en el i-ésimo casillero de la cinta en el j-ésimo paso o no, si la máquina está en un cierto estado en el j-ésimo paso o no.

Teorema de Cook-Levin (idea de demostración)

La idea es escribir una fórmula que exprese exactamente que la NDTM $\mathcal{M}$ que resuelve Π, para el input d, termina en un estado q_{si}.

Para eso, se definen variables que van a expresar si el i-ésimo casillero de la cinta contiene cierto valor en el j-ésimo paso o no, si la cabeza de la máquina está en el i-ésimo casillero de la cinta en el j-ésimo paso o no, si la máquina está en un cierto estado en el j-ésimo paso o no.

El alfabeto y la cantidad de estados e instrucciones de $\mathcal{M}$ son finitos y son constantes para el problema Π. La cantidad de pasos y posiciones de la cinta a considerar está acotada por el polinomio P que acota, dado el tamaño del input d, la cantidad máxima de pasos $P(|d|)$ en que la máquina llega al estado q_{si} si d es una instancia de respuesta "sí".

Teorema de Cook-Levin (idea de demostración)

La fórmula expresa que:

- en cada paso cada casillero de la cinta contiene exactamente un símbolo (incluyendo el blanco)

Teorema de Cook-Levin (idea de demostración)

La fórmula expresa que:

- en cada paso cada casillero de la cinta contiene exactamente un símbolo (incluyendo el blanco)
- en cada paso la cabeza está en exactamente una posición

Teorema de Cook-Levin (idea de demostración)

La fórmula expresa que:

- en cada paso cada casillero de la cinta contiene exactamente un símbolo (incluyendo el blanco)
- en cada paso la cabeza está en exactamente una posición
- en cada paso la máquina está en exactamente un estado

Teorema de Cook-Levin (idea de demostración)

La fórmula expresa que:

- en cada paso cada casillero de la cinta contiene exactamente un símbolo (incluyendo el blanco)
- en cada paso la cabeza está en exactamente una posición
- en cada paso la máquina está en exactamente un estado
- cada transición de variables de un paso al siguiente es válida de acuerdo a las tuplas que definen la máquina

Teorema de Cook-Levin (idea de demostración)

La fórmula expresa que:

- en cada paso cada casillero de la cinta contiene exactamente un símbolo (incluyendo el blanco)
- en cada paso la cabeza está en exactamente una posición
- en cada paso la máquina está en exactamente un estado
- cada transición de variables de un paso al siguiente es válida de acuerdo a las tuplas que definen la máquina
- en algún paso la máquina está en estado q_{si}.

¿Cómo se prueba que un problema es NP-completo?

A partir del Teorema de Cook, la técnica standard para probar que un problema Π es NP-completo aprovecha la transitividad de $\leq_p$, y consiste en lo siguiente:

1. Mostrar que Π está en NP.
2. Elegir un problema Π' apropiado que se sepa que es NP-completo.
3. Construir una reducción polinomial f de Π' en Π.

¿Cómo se prueba que un problema es NP-completo?

A partir del Teorema de Cook, la técnica standard para probar que un problema Π es NP-completo aprovecha la transitividad de $\leq_p$, y consiste en lo siguiente:

1. Mostrar que Π está en NP.
2. Elegir un problema Π' apropiado que se sepa que es NP-completo.
3. Construir una reducción polinomial f de Π' en Π.

La segunda condición en la definición de problema NP-completo sale usando la transitividad: sea Π'' un problema cualquiera de NP. Como Π' es NP-completo, $\Pi'' \leq_p \Pi'$. Como probamos que $\Pi' \leq_p \Pi$, resulta $\Pi'' \leq_p \Pi$.

Coloreo es NP-completo

A partir de un algoritmo de backtracking para coloreo es fácil construir una NDTM para el problema de coloreo, por lo tanto está en NP.

Coloreo es NP-completo

A partir de un algoritmo de backtracking para coloreo es fácil construir una NDTM para el problema de coloreo, por lo tanto está en NP.

Para probar que coloreo es NP-completo, vamos entonces a reducir SAT a coloreo.

Coloreo es NP-completo

A partir de un algoritmo de backtracking para coloreo es fácil construir una NDTM para el problema de coloreo, por lo tanto está en NP.

Para probar que coloreo es NP-completo, vamos entonces a reducir SAT a coloreo.

Tomemos una instancia genérica de SAT $\varphi = C_1 \wedge \cdots \wedge C_m$.
Vamos a construir un grafo G y determinar un número k de manera que φ sea satisfactible si y sólo si G se puede colorear con k-colores.

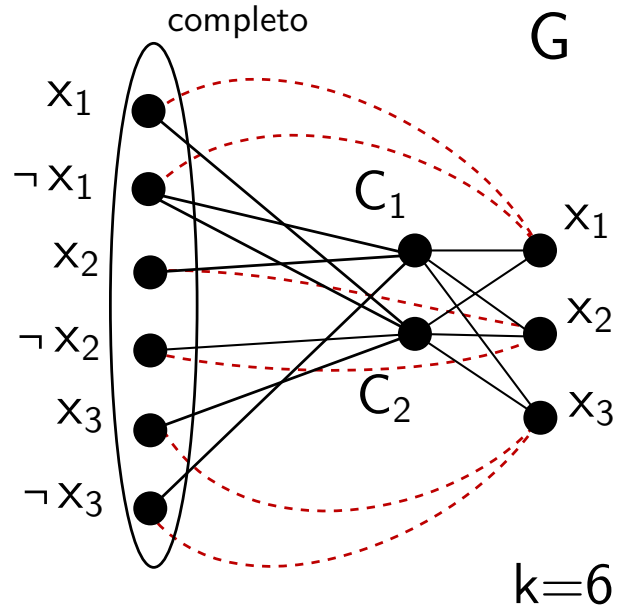
Reducción de SAT a coloreo

G tiene:

- V_1: un vértice por cada variable negada y afirmada, todos adyacentes entre si.
- V_2: un vértice por cada cláusula, adyacente a los literales de V_1 que no aparecen en la cláusula.
- V_3: otro vértice por cada variable, adyacente a todo V_2 y a los literales de V_1 correspondientes a otras variables.

k = dos veces la cantidad de variables.

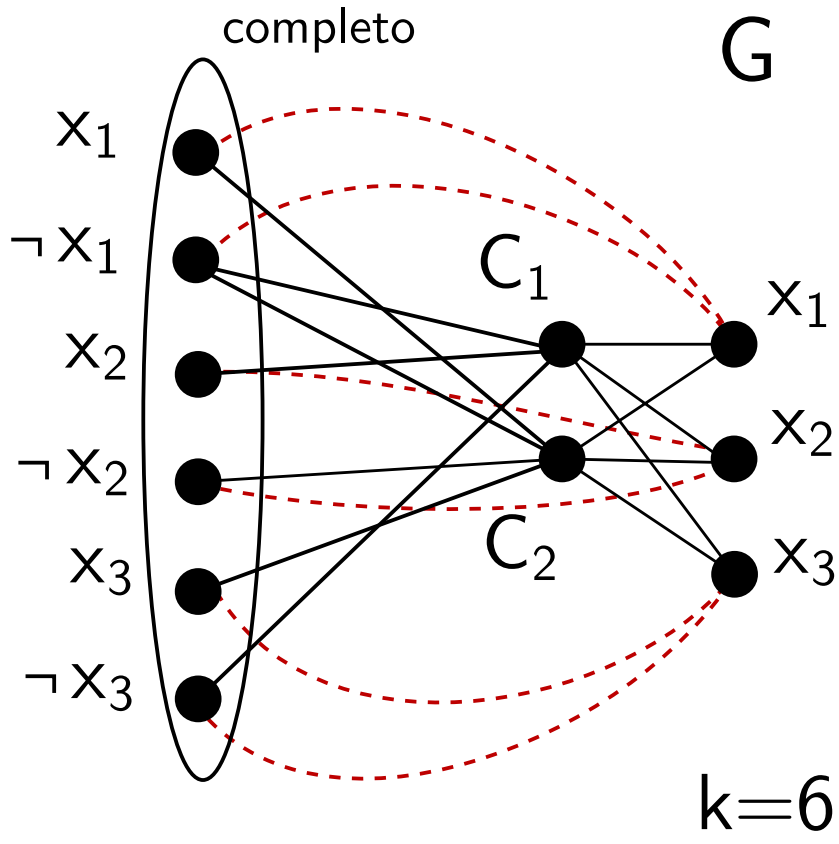
$$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3)$$



Queda como ejercicio escribir formalmente la reducción y demostrar que es una reducción polinomial de SAT a coloreo.

Reducción de SAT a coloreo

$$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3)$$



Reducción de SAT a 3-SAT

El problema 3-SAT es una variante del problema SAT, en el cual cada cláusula tiene exactamente tres literales. Como es una restricción del dominio de SAT, está en NP, y en principio es "no más difícil" que SAT.

Reducción de SAT a 3-SAT

El problema 3-SAT es una variante del problema SAT, en el cual cada cláusula tiene exactamente tres literales. Como es una restricción del dominio de SAT, está en NP, y en principio es "no más difícil" que SAT.

Para probar que 3-SAT es NP-completo, vamos entonces a reducir SAT a 3-SAT.

Reducción de SAT a 3-SAT

El problema 3-SAT es una variante del problema SAT, en el cual cada cláusula tiene exactamente tres literales. Como es una restricción del dominio de SAT, está en NP, y en principio es "no más difícil" que SAT.

Para probar que 3-SAT es NP-completo, vamos entonces a reducir SAT a 3-SAT.

Tomemos una instancia genérica de SAT $\varphi = C_1 \wedge \cdots \wedge C_m$. Vamos a reemplazar cada C_i por una conjunción de disyunciones φ'_i, donde cada disyunción tenga tres literales, y de manera que φ sea satisfactible si y sólo si $\varphi_1 \wedge \cdots \wedge \varphi_m$ lo es.

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales, queda como está.

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales, queda como está.
- C_i tiene menos de tres literales, agregamos nuevas variables como en el ejemplo:

$$(x_1 \vee \neg x_2) \rightarrow (x_1 \vee \neg x_2 \vee y) \wedge (x_1 \vee \neg x_2 \vee \neg y)$$

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales, queda como está.
- C_i tiene menos de tres literales, agregamos nuevas variables como en el ejemplo:

$$(x_1 \vee \neg x_2) \rightarrow (x_1 \vee \neg x_2 \vee y) \wedge (x_1 \vee \neg x_2 \vee \neg y)$$

- Si C_i tiene cuatro o más literales, agregamos nuevas variables como en el ejemplo:

$$(x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee \neg x_5) \rightarrow$$

$$(x_1 \vee \neg x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee \neg x_5)$$

Reducción de SAT a 3-SAT

- Si C_i tiene tres literales, queda como está.
- C_i tiene menos de tres literales, agregamos nuevas variables como en el ejemplo:

$$(x_1 \vee \neg x_2) \rightarrow (x_1 \vee \neg x_2 \vee y) \wedge (x_1 \vee \neg x_2 \vee \neg y)$$

- Si C_i tiene cuatro o más literales, agregamos nuevas variables como en el ejemplo:

$$(x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee \neg x_5) \rightarrow$$

$$(x_1 \vee \neg x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge (\neg y_2 \vee x_4 \vee \neg x_5)$$

Queda como ejercicio escribir formalmente la reducción y demostrar que es una reducción polinomial de SAT a 3-SAT.

Los 21 problemas NP-completos de Karp

En 1972, Richard Karp probó usando reducciones polinomiales que 21 problemas fundamentales de grafos y optimización combinatoria son NP-completos.

- 0-1 INTEGER PROGRAMMING (Programación lineal entera)
- CLIQUE (y conjunto independiente)
- SET PACKING (Empaquetamiento de conjuntos)
- VERTEX COVER (Recubrimiento de aristas por vértices)
- SET COVERING (Recubrimiento por conjuntos)
- FEEDBACK NODE SET (Sacar vértices para obtener un DAG)
- FEEDBACK ARC SET (Sacar arcos para obtener un DAG)
- DIRECTED HAMILTONIAN CIRCUIT (Circuito Hamiltoniano dirigido)
- UNDIRECTED HAMILTONIAN CIRCUIT (Circuito Hamiltoniano no dirigido)

Los 21 problemas NP-completos de Karp

- 3-SAT (Satisfacibilidad booleana de 3 variables por cláusula)
- CHROMATIC NUMBER (Coloreo de grafos)
- CLIQUE COVER (Recubrimiento de vértices por cliques)
- EXACT COVER (Recubrimiento por conjuntos exacto)
- HITTING SET (Conjunto de elementos que interseque una familia de conjuntos)
- STEINER TREE (Árbol de peso mínimo que conecte un conjunto de vértices)
- 3-DIMENSIONAL MATCHING (Ménage à trois)
- KNAPSACK (Problema de la mochila)
- JOB SEQUENCING (Secuenciamiento de tareas)
- PARTITION (Partición de conjuntos)
- MAX-CUT (Corte máximo)

Demostraciones de NP-completitud más recientes

Complejidad de k-coloreo en las clases de grafos sin caminos de t vértices inducidos:

k\t	4	5	6	7	8	...
3	$O(m)$ [1]	$O(n^{\alpha})$ [4]	$O(mn^{\alpha})$ [5]	$O(n^{21}(n+m))$ [6]	?	...
4	$O(m)$ [1]	P [2]	?	NPC [3]	NPC	...
5	$O(m)$ [1]	P [2]	NPC [3]	NPC	NPC	...
6	$O(m)$ [1]	P [2]	NPC	NPC	NPC	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

[1] Chvátal, 1984, Corneil, Perl, Stewart 1984.
[2] Hoàng, Kamiński, Lozin, Sawada, Shu 2010.
[3] Huang 2013.
[4] Mellin 2002.
[5] Randerath, Schiermeyer 2004.
[6] Bonomo, Chudnovsky, Macely, Schaudt, Stein, Zhong 2014.

La clase NP-difícil

Definición: Un problema de decisión Π es NP-difícil (*NP-hard*) si todo otro problema en NP se puede transformar polinomialmente a Π.

(en la práctica esta definición a veces se usa por un abuso de lenguaje también para problemas que no son de decisión y cuya versión de decisión es NP-completo)

La clase co-NP

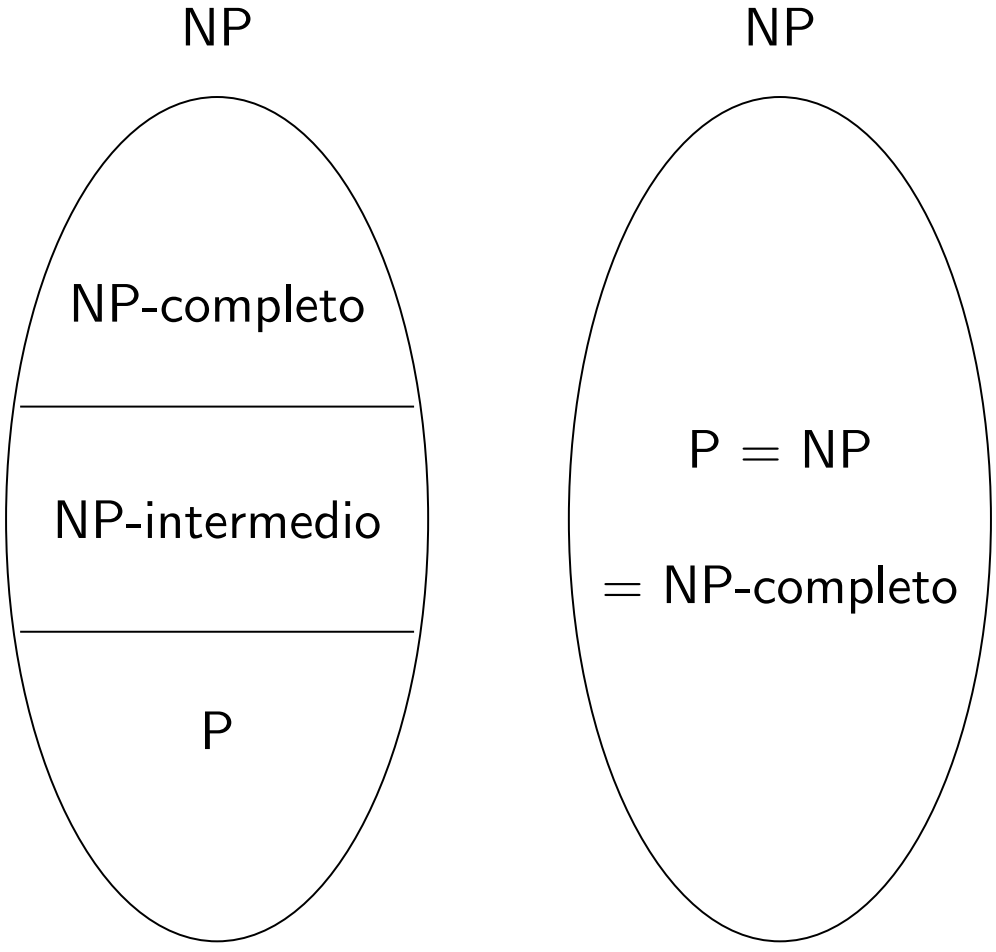
- Un problema de decisión pertenece a la clase co-NP si dada una instancia con respuesta "no" y evidencia de la misma, puede ser verificada en tiempo polinomial.
- El problema complemento de un problema de decisión Π, Π^c, es el problema de decisión que responde al complemento de la decisión de Π.
- El problema complemento tiene respuesta "no" si y sólo si Π tiene respuesta "sí".
- La clase co-NP es la clase de los problemas complemento de los problemas de la clase NP.
- La clase de los problemas polinomiales (P), está contenida también en co-NP.

Problemas abiertos

Con estas nuevas definiciones tenemos los siguientes problemas abiertos:

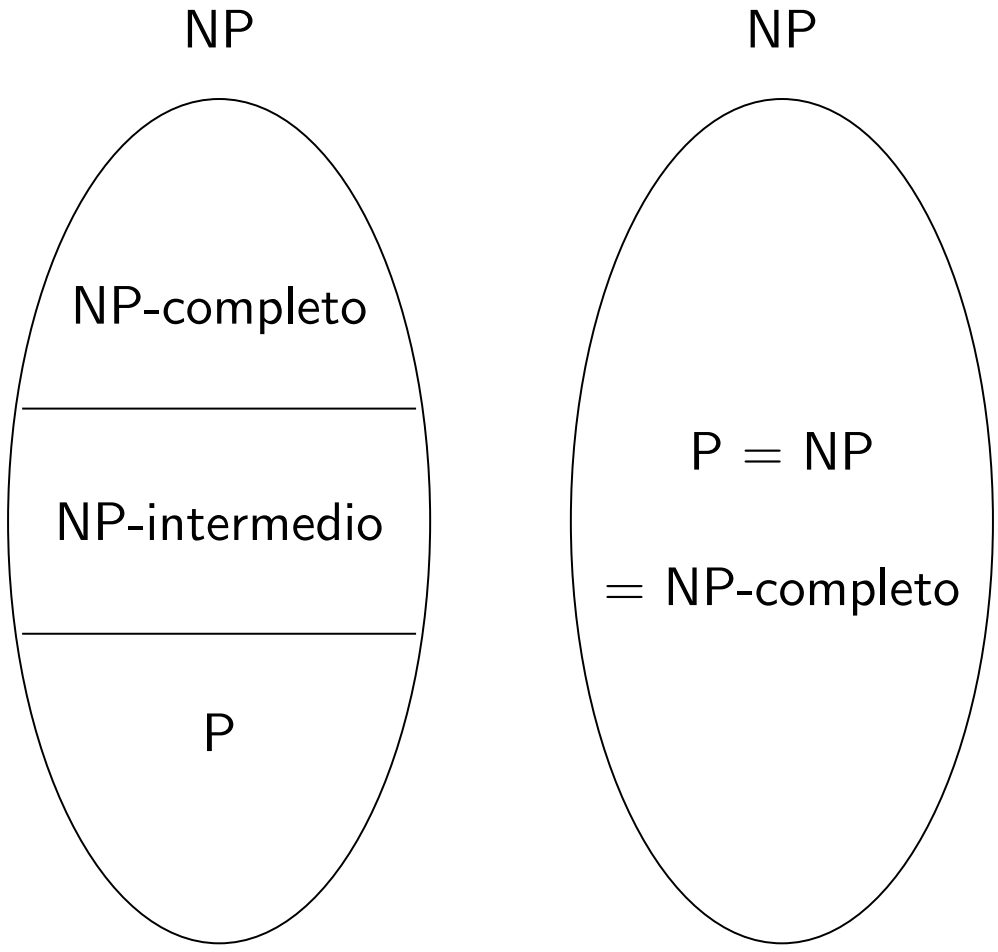
- ¿Es $P = NP$?
- ¿Es co-NP $= NP$?
- ¿Es $P =$ co-NP $\cap$ NP?

Las incógnitas...



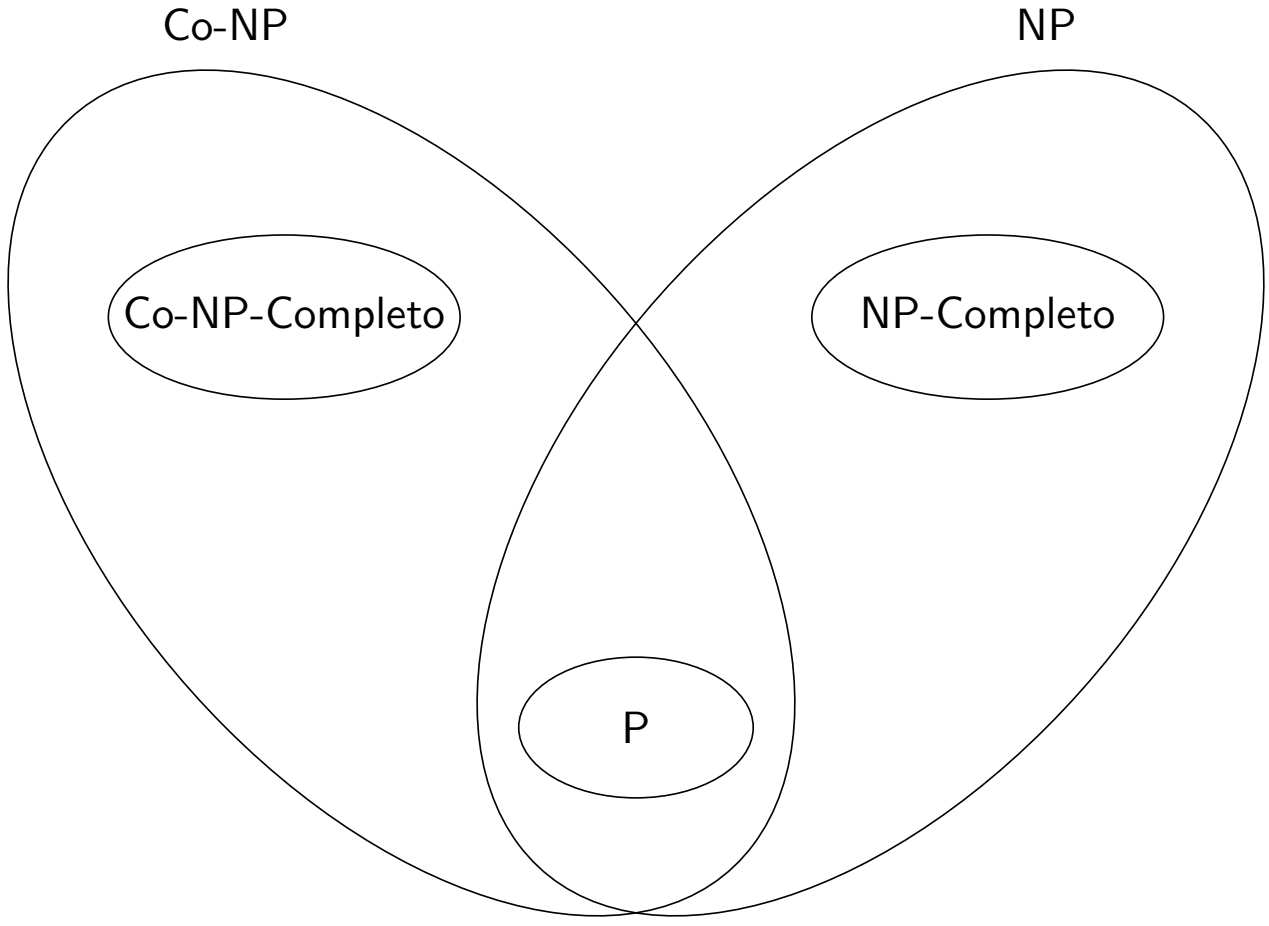
Dos mapas posibles para las clases de complejidad

Las incógnitas...



Dos mapas posibles para las clases de complejidad
R. Ladner, 1975: Si P$\neq$NP, existe la clase NP-intermedio.

Las incógnitas...



Situación si se probara que $P \neq NP$, $NP \neq co - NP$, $P \neq co - NP \cap NP$

En la práctica ...

- ¿Qué hacer ante un problema del que no sabemos en que clase está?
- ¿Qué importancia tiene saber si un problema está en P o no, desde el punto de vista teórico?
- ¿Qué importancia tiene la misma pregunta desde el punto de vista práctico?
- ¿Qué hacemos si el problema que tenemos en la práctica sabemos que es NP-completo?