Algoritmos y Estructuras de Datos III
1er cuatrimestre 2021 *(dictado a distancia)*
Paula Zabala

Teórica 10: Teoría de complejidad computacional

1. Introducción

Al comienzo de la materia, vimos cómo medir la complejidad de un algoritmo en función de la medida de la instancia, representando a ésta en una forma razonable, como puede ser en representación binaria. Y llamamos polinomial a un algoritmo que al ser ejecutado requiere una cantidad de tiempo que es una función polinomial en la medida de su entrada. Luego, estudiamos algoritmos polinomiales para algunos problemas sobre grafos. Para otros problemas, mencionamos que no se conocen algoritmos polinomiales para resolverlos.

Una idea intuitiva es que los algoritmos polinomiales son útiles en la práctica, mientras que los exponenciales no lo son. Sin embargo, ésto puede no ser verdad si el grado del polinomio es alto. Por otro lado, hay algoritmos cuyo peor caso es exponencial, pero en la práctica es muy raro que aparezca una instancia que sea peor caso y para las instancias que se presentan funciona muy bien, aun mejor que un algoritmo polinomial. Pero en este tema, casi siempre esta intuición funciona, y Edmonds en 1965 propuso la siguiente definición:

Definición 1. Un ***algoritmo eficiente*** es un algoritmo de complejidad polinomial.

El objetivo de esta última clase es analizar la dificultad inherente del ***problema*** que queremos resolver, presentando una introducción a la teoría de la complejidad computacional.

Definición 2. Un problema está ***bien resuelto*** si se conocen algoritmos eficientes para resolverlo.

Dada una instancia I de un problema de optimización Π con función objetivo f, se pueden presentar distintas versiones del problema:

- Versión de ***optimización***: Encontrar una ***solución óptima*** del problema Π para I (de valor mínimo o máximo).
- Versión de ***evaluación***: Determinar el ***valor*** de una solución óptima de Π para I.
- Versión de ***localización***: Dado un número k, determinar una ***solución factible*** de Π para I tal que $f(S) \leq k$ si el problema es de minimización (o $f(S) \geq k$ si el problema es de maximización).
- Versión de ***decisión***: Dado un número k, ¿existe una solución factible de Π para I tal que $f(S) \leq k$ si el problema es de minimización (o $f(S) \geq k$ si el problema es de maximización)?

Ejemplo 1. *TSP: Dado un grafo G con longitudes asignadas a sus aristas, las distintas versiones son:*

- *Versión de **optimización**: Determinar un circuito hamiltoniano de G de longitud mínima.*
- *Versión de **evaluación**: Determinar el valor de una solución óptima, o sea la longitud de un circuito hamiltoniano de G de longitud mínima.*
- *Versión de **localización**: Dado un número $k \in \mathbb{Z}_{\geq 0}$, determinar un circuito hamiltoniano de G de longitud menor o igual a k.*
- *Versión de **decisión**: Dado un número $k \in \mathbb{Z}_{\geq 0}$, ¿existe un circuito hamiltoniano de G de longitud menor o igual a k?*

Hay cierta relación entre la dificultad de resolver las distintas versiones de un mismo problema. En general, la versión optimización es computacionalmente la más difícil, ya que usualmente es fácil (eficiente) evaluar la función objetivo para una solución factible.

Por otro lado, la versión de decisión suele ser la ***más fácil***. Dada la respuesta de cualquiera de las otras versiones es fácil (eficiente) calcular la respuesta de esta versión.

Además, para los problemas de optimización que cumplen que el valor de una solución óptima es un entero K cuyo logaritmo está acotado por un polinomio en la medida de la entrada, la versión de evaluación puede ser resuelta ejecutando una cantidad polinomial de veces la versión de decisión, realizando búsqueda binaria sobre el parámetro k. Muchos problemas cumplen esta propiedad.

Para varios problemas, si resolvemos el problema de decisión, podemos encontrar la solución de la versión de localización resolviendo una cantidad polinomial de problemas de decisión para instancias reducidas de la instancia original. La misma relación ocurre entre las versiones de evaluación y optimización.

Ejemplo 2. *TSP: Supongamos que tenemos un grafo G y un entero $k \in \mathbb{Z}_{\geq 0}$ tales que la respuesta de la versión de decisión es **SI** y queremos resolver la versión de localización.*

Para esto seleccionamos una arista del grafo y resolvemos la versión de decisión colocando peso infinito a esa arista:

- *Si la respuesta sigue siendo **SI** significa que G tiene un circuito hamiltoniano de peso menor o igual a k que no utiliza esa arista. Dejamos el peso de esa arista en infinito.*
- *Si la respuesta pasa a ser **NO** significa que esa arista pertenece a todo circuito hamiltoniano de peso menor o igual a k. Volvemos a poner el peso original de esa arista.*

Realizamos este proceso sobre todas las aristas. Al finalizar, las aristas que quedan con peso distinto de infinito formarán un ciclo hamiltoniano de peso menor o igual a k.

Para muchos problemas de optimización combinatoria las cuatro versiones son equivalentes en el sentido que si existe un algoritmo eficiente para una de ellas, entonces existe para todas.

La clasificación y el estudio de los problemas dentro de esta teoría se realiza sobre problemas de decisión.

Definición 3. Un ***problemas de decisión*** es un problema cuya respuesta es ***SI*** o ***NO***.

Por un lado, ésto permite uniformar el estudio, ya que hay problemas, como saber si un grafo es hamiltoniano o satisfabilidad, que no tienen versión de optimización. Por otro lado, un resultado negativo para la versión de decisión, por lo que vimos anteriormente, se traslada a un resultado negativo a la versión de optimización.

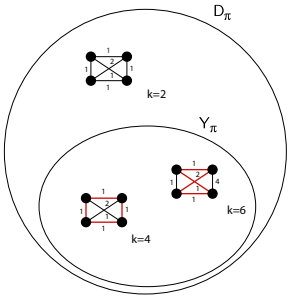
Hasta ahora estuvimos trabajando con instancias de un problema y nos entendimos, pero ahora definiremos formalmente este concepto.

Definición 4.

- Una ***instancia*** de un problema es una especificación de sus parámetros.
- Un problema de decisión Π tiene asociado:

- un conjunto D_Π de instancias
- y un subconjunto $Y_\Pi \subseteq D_\Pi$ de instancias cuya respuesta es SI.

Ejemplo 3. *TSP: Dado un grafo completo con peso en las aristas y un número $k \in \mathbb{Z}_{\geq 0}$, ¿existe un circuito Hamiltoniano de longitud a lo sumo k?*



Nos interesa reconocer los problemas computacionalmente difíciles.

Definición 5. Un problema es ***intratable*** si no puede ser resuelto por algún algoritmo eficiente.

Un problema puede ser intratable por distintos motivos:

- El problema requiere una repuesta de longitud exponencial (ejemplo: pedir todos los circuitos hamiltonianos de longitud a lo sumo k).
- El problema es ***indecidible*** (ejemplo: problema de la parada).
- El problema es decidible pero no se conocen algoritmos polinomiales que lo resuelvan (no se sabe si es intratable o no).

2. Problema de satisfabilidad (SAT)

Una variable booleana o lógica es una variable que puede tomar valor verdadero (V) o falso (F). Un literal es una variable booleana o su negación y una expresión o fórmula booleana está formada por variables, conjunciones, disyunciones y negaciones. Una cláusula es una disyunción de literales y una fórmula está en forma normal conjuntiva (FNC) si es una conjunción de cláusulas. Toda fórmula lógica puede ser expresada en FNC.

Una fórmula se dice satisfacible si existe una asignación de valores de verdad a sus variables tal que la haga verdadera.

Ejemplo 4. *Dados $X = \{x_1, x_2, x_3\}$ un conjunto de variables booleanas y las fórmulas en FNC:*

- $\varphi_1 = (x_1 \vee \neg x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_3)$ *es satisfacible, ya que asignando los valores de verdad $x_1 = x_2 = V$ y $x_3 = F$, φ_1 es verdadera.*
- $\varphi_2 = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_2) \wedge (\neg x_3)$ *no es satisfacible, porque no existe asignación de valores de verdad a las variables de X que hagan a φ_2 verdadera.*

Veremos que SAT tiene un rol central en el estudio de la teoría de la complejidad computacional. Su definión es la siguiente.

Definición 6. ***Problema de satisfabilidad (SAT):*** Dado un conjunto de cláusulas $C_1, \ldots, C_m$ formadas por literales basados en las variables booleanas $X = \{x_1, \ldots, x_n\}$, ¿la expresión $\varphi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ es satisfacible?

3. Modelos de cómputo

Los modelos de cómputo son modelos abstractos para expresar cualquier algoritmo. Describen formalmente cómo se calcula la salida en función de la entrada. Como lo hicimos en la primera clase, un modelo de cómputo brinda el marco formal, independiente de la implementación particular, para calcular la complejidad de un algoritmo. Dos de los más utilizados son:

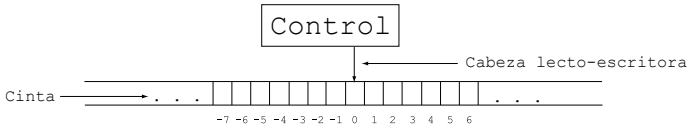
- Máquina de Turing (1937, Alan Turing)
- Máquinas de Acceso Random - RAM (1974, Aho, Hopcroft y Ullman).

Estos dos modelos son polinomialmente equivalentes. Es decir, se puede simular uno a otro con un costo polinomial. Por practicidad e historia, al comienzo de la materia utilizamos las máquinas RAM para formalizar el cálculo de la complejidad de un algoritmo y ahora utilizaremos las máquinas de Turing para estudiar una introducción a la Teoría de la complejidad computacional.

3.1. Máquina de Turing Determinística

Una ***máquina de Turing*** consiste de:

- Una cabeza lecto-escritora.
- Una cinta infinita con el siguiente esquema:



- La cabeza lectora se puede mover en ambas direcciones: $M = \{+1, -1\}$ son los movimientos de la cabeza a derecha $(+1)$ o izquierda (-1).
- Un conjunto finito de estados, Q.
- Hay una celda distinguida, la celda 0, que define la celda incial.
- Un alfabeto finito Σ y un símbolo especial $*$ que indica ***blanco***, definiendo $\Gamma = \Sigma \cup \{*\}$.
- Sobre la cinta está escrita la entrada, que es un string de símbolos de Σ y el resto de las celdas tiene $*$ (blancos).
- Hay un estado distinguido, $q_0 \in Q$, que define el estado inicial.
- Y un conjunto de estados finales, $Q_f \subseteq Q$ (q_{si} y q_{no} para problemas de decisión)

Definición 7.

- Una ***instrucción*** en una MT es una quíntupla $S \subseteq Q \times \Gamma \times Q \times \Gamma \times M$. La quíntupla $(q_i, s_h, q_j, s_k, +1)$ se interpreta como:

Si la máquina está en el estado q_i y la cabeza lee s_h, entonces escribe s_k, se mueve a la derecha y pasa al estado q_j.

- Se define un ***programa*** en una MT como un conjunto finito de instrucciones.
- Una MT es ***determinística***, MTD, si para todo par (q_i, s_h) existe en el programa a lo sumo una quíntupla que comienza con ese par.

El funcionamiento de una MT es el siguiente:

- Arranque
  - máquina posicionada en el estado distinguido q_0, estado inicial
  - cabeza lectora-escritora ubicada en la celda inicial (0) de la cinta.
- Transiciones
  - si la máquina está en el estado q_i y lee en la cinta s_h, busca en la tabla de instrucciones un quíntupla que comience con el par (q_i, s_j).
  - si existe esta quíntupla, (q_i, s_h, q_j, s_k, m), la máquina pasa al estado q_j, escribe en la cinta el símbolo s_k y se mueve a derecha o izquieda según el valor de m.
- Terminación
  - cuando no se puede inferir nuevas acciones para seguir: no existe la quíntupla que comienza con (q_i, s_h)
  - cuando se alcanza un estado final: si el estado final es de SI, entonces la respuesta es ***SI***, caso contrario la respuesta es ***NO***.

Definición 8.

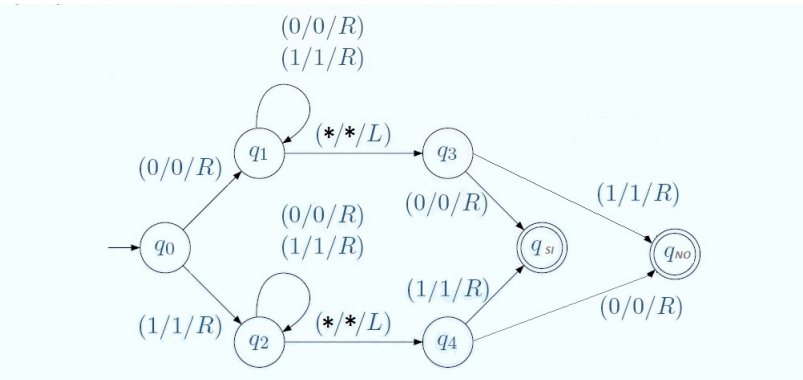
- Una máquina M resuelve el problema Π si para toda instancia alcanza un estado final y responde de forma correcta (o sea, termina en un estado final correcto).
- La complejidad de una MTD está dada por la cantidad de movimientos de la cabeza, desde el estado inicial hasta alcanzar un estado final, en función del tamaño de la entrada.

$$T_M(n) = \text{máx}\{m \text{ tq } x \in D_\Pi, |x| = n \text{ y } M \text{ con entrada } x \text{ hace } m \text{ movimientos}\}$$

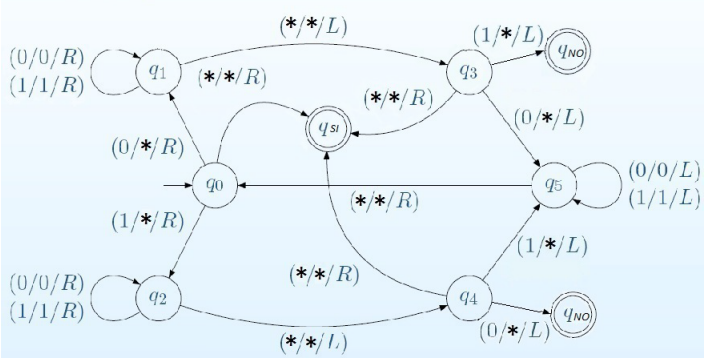
Existen otros modelos de computadoras determinísticas (máquina de Turing con varias cintas, Random Access Machines, etc.) pero puede probarse que son equivalentes en términos de la polinomialidad de los problemas a la MTD.

Una forma útil de expresar una MD es mediante un gráfico, donde los estados se representan mediante nodos y las transiciones mediante los enlaces. Veamos dos ejempplos.

Ejemplo 5. *Dada una palabra sobre* $\{0, 1\}$*, ¿comienza y termina con el mismo símbolo?*



Ejemplo 6. *Dada una palabra sobre* $\{0, 1\}$*, ¿es palíndromo?*



3.2. Máquinas de Turing no-determinísticas (MTND)

Una ***máquina de Turing no-determinística***, MTND, es una generalización de una MTD. Tiene los mismos componentes que vimos para una máquina de Turing determinística, pero no se pide unicidad de la quíntupla que comienza con cualquier par (q_i, s_j).

Las instrucciones dejan de ser quíntuplas para pasar a ser un mapeo multivaluado. Un programa correspondiente 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 permite la ejecución en paralelo de las distintas alternativas. Cuando una MTND llega a un punto en su ejecución donde son posibles múltiples alternativas, la MTND examina todas las alternativas en paralelo, en lugar de hacerlo secuencialmente como una MTD. Podemos pensarlo como que se abre en k copias cuando se encuentra con k alternativas. Cada copia continua su ejecución independientemente. Si una copia acepta la entrada, todas las copias paran.

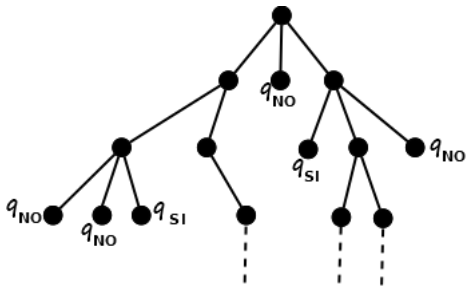
Una interpretación equivalente es que cuando se presentan múltiples alternativas, la MTND selecciona la ***alternativa correcta***.

Definición 9. Una MTND resuelve el problema de decisión Π si:

- existe una secuencia de alternativas que lleva a un estado de respuesta ***SI*** si, y sólo si, la respuesta es ***SI***, o bien
- alguna de las copias se detiene en un estado de respuesta ***SI*** si, y sólo si, la respuesta es ***SI***.

Ésto es equivalente a: Para toda instancia de Y_Π existe una rama que llega a un estado final q_{si} y para toda instancia en $D_\Pi \setminus Y_\Pi$ ninguna rama llega a un estado final q_{si}.

Podemos interpretar la ejecución de una MTND como un árbol de alternativas. Para una instancia de Y_Π será:



Definición 10.

- La complejidad temporal de una MTND M se define como el máximo número de pasos que toma como mínimo reconocer una instancia de Y_Π en función de su tamaño:

$$T_M(n) = \text{máx}\{m \text{ tq } x \in Y_\Pi, |x| = n \text{ y alguna de las ramas de } M \text{ termina en estado } q_{si} \text{ en } m \text{ movimientos cuando la entrada es } x\}$$

- Una MTND M es ***polinomial*** para Π cuando $T_M(n)$ es un función polinomial.

4. Las clases P y NP

Definición 11. Un problema de decisión Π pertenece a la clase **P** (polinomial) si existe una MTD de complejidad polinomial que lo resuelve:

$$\mathbf{P} = \{\Pi \text{ tq } \exists M \text{ MTD tq } M \text{ resuelve } \Pi \text{ y } T_M(n) \in O(p(n)) \text{ para algún polinomio } p\}$$

Ésto es equivalente a: existe un algoritmo polinomial que lo resuelve.

Ejemplos de problemas en **P** (en su versión de decisión):

- Grafo conexo.
- Grafo bipartito.
- Árbol generador mínimo.
- Camino mínimo entre un par de nodos.
- Matching máximo.
- Grafo euleriano.

Una clase más general de problemas de decisión es la siguiente:

Definición 12. Un problema de decisión Π pertenece a la clase **NP** (polinomial no-determinístico) si las instancias de Π con respuesta ***SI*** son reconocidas por una MTND polinomial.

Equivalentemente, dada una instancia de Π con respuesta ***SI*** se puede dar un certificado de longitud polinomial que garantiza que la respuesta es ***SI***, y esta garantía puede ser verificada en tiempo polinomial.

Esta clase es de interés porque incluye a la gran mayoría de los problemas de optimización combinatoria en su versión decisión.

Ejemplo 7. *Dado un grafo $G = (V, X)$, un **conjunto independiente** de vértices de G, es un conjunto de vértices $I \subseteq V$ tal que para toda arista $e \in X$, e es incidente a lo sumo a un vértice $v \in I$. El problema de conjunto independiente máximo en su versión de decisión es* **NP**.

CONJUNTO INDEPENDIENTE: Dados un grafo $G = (V, X)$ y $k \in \mathbb{N}$, ¿G tiene un conjunto independiente de tamaño mayor o igual a k?

*Para una instancia con respuesta **SI**, podemos exponer $S \subseteq V$ conjunto independiente de G tal que $|S| \geq k$. Es posible chequear polinomialmente que S cumple estas dos propiedades: ser conjunto independiente de G y tener*

cardinal mayor o igual a k.

Esto demuestra que CONJUNTO INDEPENDIENTE pertenece a la clase **NP**.

Ejemplo 8. *SAT es* **NP**:

Una instancia de ***SI*** *de SAT es un conjunto de cláusulas $C_1, \ldots, C_m$ satisfacible. Ésto es, existe una asignación de valores de verdad a las variables booleanas intervinientes en las cláusulas que hacen verdadero el valor de verdad de la expresión $C_1 \wedge C_2 \wedge \ldots \wedge C_m$.*

El certificado que podemos mostrar es una asignación de valores de verdad a las variables que haga verdadera a la expresión $C_1 \wedge C_2 \wedge \ldots \wedge C_m$. Como es posible verificar en tiempo polinomial que esta expresión es verdad con esos valores de las variables, demuestra que $SAT \in$ **NP**.

Ejemplo 9. *El problema de clique máxima en su versión de decisión es* **NP**.

CLIQUE: Dado un grafo $G = (V, X)$ y $k \in \mathbb{N}$, ¿G tiene una clique de tamaño mayor o igual a k?

Dada una instancia de ***SI****, ésto es un grafo $G = (V, X)$ que tiene una clique de tamaño mayor o igual a k, podemos verificar polinomialmente que un conjunto de vértices C cumple que:*

- *$C \subseteq V$*
- *C induce una clique en G*
- *$|C| \geq k$*

La existencia de este algoritmo polinomial demuestra que CLIQUE $\in$ **NP**.

Ejemplo 10. *CIRCUITO HAMILTONAINO está en* **NP**:

La evidencia que soporta una respuesta positiva es un ciclo hamiltoniano de G. Dado un lista de vértices, se puede chequear polinomialmente si define un ciclo hamiltoniano.

Ejemplo 11. *TSP está en* **NP**:

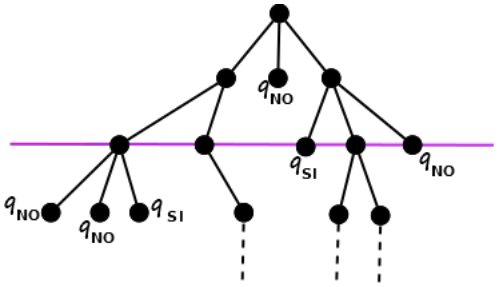
Dada una instancia de TSP, G y $k \in \mathbb{Z}_{\geq 0}$, la evidencia que soporta una respuesta positiva es un ciclo hamiltoniano de G con peso menor o igual a k. Dada una lista de vértices, se puede chequear polinomialmente si define un ciclo hamiltoniano y que la suma de los pesos de las aristas respectivas es menor o igual a k.

Un resultado interesante sobre los problemas de la clase **NP** es el siguiente:

Lema 1. *Si Π es un problema de decisión que pertence a la clase* **NP**, *entonces Π puede ser resuelto por un algoritmo determinístico en tiempo exponencial respecto del tamaño de la entrada.*

Demostración. Como $\Pi \in$ **NP**, existe una MTND polinomial M que lo resuelve. Sea $T_M(n)$ el polinomio que define la complejidad de esta MTND.

Para resolver Π de forma determinística secuencial podemos recorrer cada rama del árbol de ejecución de M hasta profundidad $T_M(n)$ o llegar a un estado final, lo que ocurra primero. Si en alguna rama se llega a un estado q_{SI} la respuesta será ***SI***, caso contrario será ***NO***. Es decir, es posible cortar el árbol de ejecución a altura $T_M(n)$ para obtener la respuesta correcta.



La cantidad de alternativas en las que se puede abrir la MTND está acotada, ya que la cantidad de ternas está acotada por ser el alfabeto y la cantidad de estados finitos. Si c es una cota de la cantidad de ternas, en el nivel r del árbol de ejecución, habrá, a lo sumo, c^r copias en ejecución paralelamente.

Por lo tanto, la implementación secuencial del algoritmo recorrerá, a lo sumo, $\sum_{i=0}^{T_M(n)} c^i$ nodos (cantidad de movimientos).

■

Los primeros hechos que podemos observar sobre la relación entre estas clases son:

- **P** $\subseteq$ **NP**: Porque una MTD es un caso particular de MTND.
- ***Problema abierto:*** ¿Es **P** = **NP**?
  - Todavía no se demostró que exista un problema en NP\P.
  - Mientras tanto, se estudian clases de complejidad ***relativa***, es decir, que establecen orden de dificultad entre problemas.

5. Transformaciones polinomiales

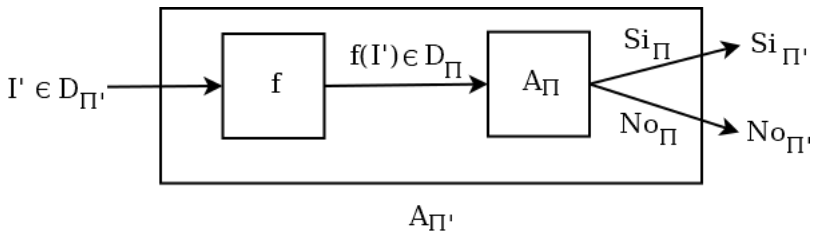
Imaginemos que tenemos un algoritmo A_Π para resolver un problema Π y ahora queremos resolver otro problema Π'. ¿Habrá forma de que podamos aprovechar A_Π?

Para ésto, deberíamos ser capaces de, dada una instancia I' del problema Π', poder transformarla en una instancia I del problema Π, tal que la respuesta para I sea ***SI*** si, y sólo si, la respuesta para I' es ***SI***.

Lo que necesitamos es una transformación $f : D_{\Pi'} \to D_\Pi$, tal que

$$I' \in Y_{\Pi'} \iff f(I') \in Y_\Pi.$$

Entonces, un algoritmo posible para resolver Π' podría ser $A_{\Pi'}(I') = A_\Pi(f(I'))$, donde I' es una instancia de Π'.



Si la función f y el algoritmo A_Π son polinomiales, el algoritmo $A_{\Pi'}$ resultará polinomial.

Definamos esta transformación formalmente:

Definición 13.

- Una ***transformación o reducción polinomial*** de un problema de decisión Π' a uno Π es una función polinomial que transforma una instancia I' de Π' en una instancia I de Π tal que I' tiene respuesta ***SI*** para Π' si, y sólo si, I tiene respuesta ***SI*** para Π:

$$I' \in Y_{\Pi'} \iff f(I') \in Y_\Pi$$

- El problema de decisión Π' se ***reduce polinomialmente*** a otro problema de decisión Π, $\Pi' \leq_p \Pi$, si existe una transformación polinomial de Π' a Π.

Proposición 1. *La reducción polinomial es una relación transitiva: Si P_1, P_2 y P_3 son problemas de decisión tales que $\Pi_1 \leq_p \Pi_2$ y $\Pi_2 \leq_p \Pi_3$, entonces $\Pi_1 \leq_p \Pi_3$.*

Demostración. Sea $f : D_{\Pi_1} \to D_{\Pi_2}$ una reducción polinomial de P_1 a P_2 y $g : D_{\Pi_2} \to D_{\Pi_3}$ de P_2 a P_3.

Entonces

$$I_1 \in Y_{\Pi_1} \iff f(I_1) \in Y_{\Pi_2} \text{ y } I_2 \in Y_{\Pi_2} \iff g(I_2) \in Y_{\Pi_3}.$$

Por lo tanto, para toda $I_1 \in D_{\Pi_1}$ obtenemos que:

$$I_1 \in Y_{\Pi_1} \iff g(f(I_1)) \in Y_{\Pi_3}.$$

Además, la composición de dos funciones polinomiales resulta en una función polinomial. Resultando que $\Pi_1 \leq_p \Pi_2$. ■

6. Clase NP-completo

Este concepto es el centro de la teoría de la complejidad computacional. Intuitivamente, agrupa a los problemas de decisión que son los más difíciles de la clase **NP**.

Los problemas de la clase **NP-Completo** tienen dos propiedades sumamente importantes:

- No se conocen algoritmos polinomiales para resolverlos.
- Si existe un algoritmo polinomial para uno de ellos, existen algoritmos polinomiales para todos ellos.

Una creencia es que los problemas de esta clase son intratables desde el punto de vista computacional, por lo que no existirían algoritmos eficientes para resolverlos. Pero ésto no está demostrado y podría no ser verdadero.

Formalmente:

Definición 14. Un problema de decisión Π es **NP-completo** si:

1. $\Pi \in$ **NP**
2. $\forall \bar{\Pi} \in$ **NP**, $\bar{\Pi} \leq_p \Pi$

Se define la clase ***NP-difícil*** como los problemas que cumplen la condición 2 (al menos tan "difícil" como todos los problemas de **NP**).

SAT fue el primer problema que se demostró que pertenece a la clase **NP-completo**, dando origen a esta clase.

Teorema 1 (Teorema de Cook(1971) - Levin(1973))**.** *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 MTND que resuelve Π, genera en tiempo polinomial una fórmula lógica $\varphi_{\Pi,d}$ en forma normal (conjunción de disyunciones) tal que $d \in Y_\Pi$ si, y sólo si, $\varphi_{\Pi,d}$ es satisfactible.

Usando la transitividad de las reducciones polinomiales, a partir de este primer resultado podemos probar que otros problemas son **NP-completos**.

Si Π es un problema de decisión, podemos probar que $\Pi \in$ **NP-completo** encontrando otro problema Π_1 que ya sabemos que es **NP-completo** y demostrando que:

1. $\Pi \in$ **NP**

2. $\Pi_1 \leq_p \Pi$

La segunda condición en la definición de problema NP-completo se deriva de la transitividad:

Sea Π' un problema cualquiera de NP. Como Π_1 es NP-completo, sabemos que $\Pi' \leq_p \Pi_1$. Como ahora probamos que $\Pi_1 \leq_p \Pi$, resulta, por transitividad de las reducciones polinomiales, $\Pi' \leq_p \Pi$.

Desde 1971, se ha probado la NP-completitud de muchos problemas usando el método anterior. A partir del Teorema de Cook-Levin, Richard Karp demostró en 1972 que otros 21 problemas son NP-completos. Actualmente se conocen más de 3.000 problemas pertenecientes a esta clase.

Ejemplo 12. *CLIQUE es* **NP-completo**.

Para demostrar que CLIQUE es **NP-completo**, *alcanza con probar que:*

1. *CLIQUE* $\in$ **NP**.
2. *Para algún problema* $\Pi \in$ **NP-completo**, $\Pi \leq_p$ *CLIQUE.*

En el ejemplo 9 ya probamos 1.

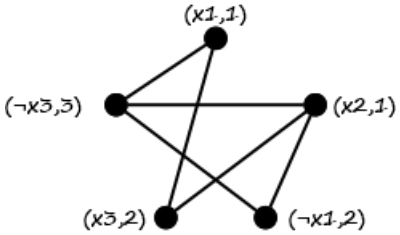
Para probar la segunda condición utilizaremos SAT y demostraremos que SAT $\leq_p$ *CLIQUE:*

Dada una intancia de SAT, I_S*, tenemos que transformarla polinomialmente en una instancia de CLIQUE,* I_C*, tal que CLIQUE responda SI para* I_C *si, y sólo si, SAT responde SI para* I_S*.*

Para ésto, tomemos una instancia genérica de SAT, I_S*, con* k_S *cláusulas.*

Definimos $k_C = k_S$ y $G = (V, X)$ de la siguiente manera. Para cada literal x en la cláusula i, ponemos un vértice $(x, i) \in V$ y $((x, i), (y, j)) \in X$ si, y sólo si, $x \neq \bar{y}$ y $i \neq j$.

$$\varphi = C_1 \wedge C_2 \wedge C_3 = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3)$$



Tenemos que ver que I_S es satisfcible si, y sólos si, G tiene una clique de tamaño mayor o igual a k.

Si I_S es satisfacible, existe un conjunto de literales que hacen verdaderas las k_S cláusulas simultáneamente. Tomando para cada cláusula un literal de este conjunto, representarán distintos vértices en G todos adyacentes entre sí (por ser de distintas cláusulas), formando una clique de tamaño $k_S = k_C$ en G.

Por otro lado, si H es una clique de G de tamaño mayor o igual a k_C, cada vértice de H representará un literal de una cláusula distinta que pueden ser simultáneamente verdaderos por no ser complementarios. Asignándole valor verdadero a estos literales, I_S toma valor verdadero. Por lo tanto, I_S es satisfacible.

Ejemplo 13. *3-SAT es* **NP-completo**.

El problema 3-SAT es un caso particular del problema SAT, en el cual se pide que cada cláusula tenga exáctamente tres literales.

1. *Como 3-SAT es un caso particular de SAT y ya sabemos que SAT $\in$* **NP**, *podemos decir que 3-SAT está en* **NP**.

2. *Para probar que 3-SAT es* **NP-completo**, *vamos a reducir polinomialmente SAT a 3-SAT.*

   Tomemos una instancia genérica de SAT, $\varphi = C_1 \wedge \cdots \wedge C_m$ sobre las variables booleanas $x_1, \ldots, x_n$. Vamos a reemplazar cada C_i por una conjunción de disyunciones φ'_i, donde cada disyunción tenga tres literales, de manera que φ sea satisfactible si, y sólo si, $\varphi' = \varphi'_1 \wedge \cdots \wedge \varphi'_m$ lo es.

   - *Si C_i tiene tres literales:*
   $$\varphi'_i = C_i.$$
   - *C_i tiene dos literales, a_1 y a_2, agregamos una variable nueva y y definimos:*
   $$C_i = (a_1 \vee a_2) \rightarrow \varphi'_i = (a_1 \vee a_2 \vee y^i) \wedge (a_1 \vee a_2 \vee \neg y^i).$$
   - *Si C_i tiene $k \geq 4$ literales, agregamos $k - 3$ variables nuevas:*
   $$C_i = (a_1 \vee a_2 \vee \ldots \vee a_r \vee \ldots \vee a_{k-1} \vee a_k) \rightarrow$$
   $$\varphi'_i = (a_1 \vee a_2 \vee y^i_1) \wedge \ldots \wedge (\neg y^i_{r-2} \vee a_r \vee y^i_{r-1}) \wedge \ldots \wedge (\neg y^i_{k-3} \vee a_{k-1} \vee a_k)$$
   Por ejemplo: $C_i = (a_1 \vee a_2 \vee a_3 \vee a_4 \vee a_5) \rightarrow \varphi'_i = (a_1 \vee a_2 \vee y^i_1) \wedge (\neg y^i_1 \vee a_3 \vee y^i_2) \wedge (\neg y^i_2 \vee a_4 \vee a_5)$

Falta ver que φ es satisfactible si, y sólo si, $\varphi' = \varphi'_1 \wedge \cdots \wedge \varphi'_m$ lo es.

Para ver esto, supongamos primero que φ es satisfacible. Sea $x_1^*, \ldots, x_n^*$ valores de las variables $x_1, \ldots, x_n$ que hacen verdadera φ.

Entonces por lo menos un literal a_j^i de cada cláusula C_i tiene que ser verdadero (si hay más de uno elegimos el primero por ejemplo). Para ver que φ' es satisfacible, mantenemos el valor de las variables originales $x_j = x_j^*$ para $j = 1, \ldots, n$. Si el literal a_j^i es verdadero, fijamos $y_1^i, \ldots, y_{j-2}^i$ en verdadero y $y_{j-1}^i, \ldots, y_{k-3}^i$ en falso. Esto hará a la cláusula φ'_i verdadera, haciendo a φ verdadera.

Ahora supongamos que $\varphi' = \varphi'_1 \wedge \cdots \wedge \varphi'_m$ es verdadera. Al menos uno de los literales a_j^i de la conjunción de cláusulas φ'_i debe ser verdadero, de lo contrario y_1^i debe ser verdadera, pero eso impone que y_2^i sea verdadera, y así siguiendo. Pero ésto hará que la última cláusula φ'_m sea falsa, contradiciendo que φ' es verdadera. Ésto asegura que tomando esos valores de las variables x_j, la cláusula C_i es verdadera.

Ejemplo 14. *COLOREO es* **NP-completo**.

1. *Es fácil probar que COLOREO es* **NP**.

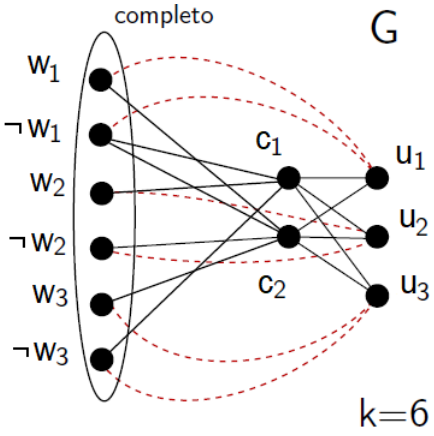
2. *Para probar que COLOREO es* **NP-completo**, *vamos a reducir polinomialmente SAT a COLOREO.*

   Tomemos una instancia genérica de SAT, $\varphi = C_1 \wedge \cdots \wedge C_m$ sobre las variables $x_1, \ldots, x_n$. 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.

   Definimos k = dos veces la cantidad de variables de φ y $G = (V_1 \cup V_2 \cup V_3, X)$ como:

   - *V_1: dos vértice w_i y $\neg w_i$ por cada variable x_i (representando a todos los posibles literales), todos adyacentes entre si, formando un K_k.*
   - *V_2: un vértice c_j por cada cláusula C_j, adyacente a los literales de V_1 que no aparecen en la cláusula.*
   - *V_3: un vértice u_i por cada variable x_i, adyacente a todos los vértices de V_2 y a los literales de V_1 correspondientes a otras variables (u_i no adyacente a w_i ni a $\neg w_i$).*

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



Falta ver que φ es satisfactible si, y sólo si, G es k-coloreable.

Supongamos que φ es satisfactible. Sea $x_1^*, \ldots, x_n^*$ valores de las variables $x_1, \ldots, x_n$ que hacen verdadera φ. Pintamos a G con k colores de la siguiente forma:

- *Para cada vértice de V_1 obviamente utilizamos un color distinto.*
- *Al vértice $c_j \in V_2$ lo pintamos del color del vértice de V_1 correspondiente a un literal que la hace verdadera.*
- *Si $x_i^* = V$, pintamos a $u_i \in V_3$ con el mismo color que $\neg w_i$, y si $x_i^* = F$, con el mismo color que w_i.*

Obtenemos así un k-coloreo de G.

Consideremos ahora que G es k-coloreable. En un k-coloreo de G, cada uno de los k colores se utiliza exáctamente una vez en los vértices de V_1. Un vértice $u_i \in V_3$ tendrá el mismo color que w_i o de $\neg w_i$, ya que es adyacente al resto de los vértices de V_1. El color del vértice c_j será el mismo que tiene algún vértice de V_1 representando a un literal de la cláusula C_j (porque es adyacente al resto de los vértices de V_1) y distinto a todos los colores de los vértices de V_3.

Si u_i tiene el mismo color que w_i, le asignamos valor F a la variable x_i, mientras que si tiene el mismo color que $\neg w_i$, le asignamos valor V. Entonces, el vértice c_j comparte color con un literal que es verdadero en esta asignación, haciendo este literal verdadera a la cláusula C_j. Por lo tanto esta asignación hace verdadera a φ, demostrando que φ es satisfacible.

Ejemplo 15. *CONJUNTO INDEPENDIENTE es* **NP-completo**.

1. *Es fácil probar que CONJUNTO INDEPENDIENTE es* **NP**.
2. *Para probar que es* **NP-difícil**, *vamos a reducir polinomialmente CLIQUE a CONJUNTO INDEPENDIENTE.*

Tomemos una instancia genérica de CLIQUE, I_C, esto es un grafo G y $k \in \mathbb{N}$ y definamos la instancia I_{CI} de CONJUNTO INDEPENDIENTE, $\bar{G}$ y k.

Como una clique S en un grafo G es un conjunto independiente en $\bar{G}$, se cumple que G tiene una clique de tamaño mayor o igual a k si, y sólo si, $\bar{G}$ tiene un conjunto independiente de tamaño mayor o igual a k.

Ésto muestra que CLIQUE responde que SI para I_C si, y sólo si, CONJUNTO INDEPENDIENTE responde SI para I_{CI}.

Ejemplo 16. *Dado un grafo $G = (V, X)$, un* ***recubrimiento de las aristas*** *de G, es un conjunto $R_a \subseteq V$ de vértices tal que para todo $e \in X$, e es incidente al menos a un vértice $v \in R_n$. El problema de RECUBRIMIENTO DE ARISTAS en su versión de decisión es: dados un grafo $G = (V, X)$ y $k \in N$, ¿G tiene un recubrimiento de aristas de tamaño menor o igual a k?*

NP-completo.

1. *Es fácil probar que RECUBRIMIENTO DE ARISTAS es* **NP**.

2. *Para probar que es* **NP-completo**, *vamos a reducir polinomialmente CONJUNTO INDEPENDIENTE a RECUBRIMIENTO DE ARISTAS.*

   Tomamos una instancia genérica de CONJUNTO INDEPENDIENTE, $G = (V, X)$ de n vértices y $k \in \mathbb{N}$, $k \leq n$ y la transformamos a la instancia de RECUBRIMIENTO DE ARISTAS, G y $k' = n - k$.

   Dado $G = (V, X)$, S es conjunto independiente si, y sólo si, $V \setminus S$ es recubrimiento de aristas. Esto asegura que G tendrá un conjunto independiente de tamaño mayor o igual a k si, y sólo si, G tiene un recubrimiento de aristas de tamaño menor o igual a k'.

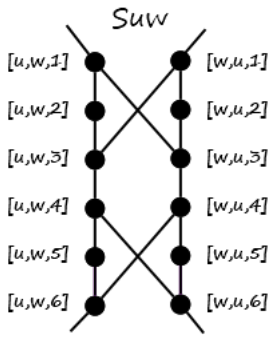
Ejemplo 17. *CIRCUITO HAMILTONIANO es* **NP-completo** *(optativo).*

1. *En el Ejemplo 10 mostramos que este problema es* **NP**.

2. *Para probar que es* **NP-completo**, *vamos a reducir polinomialmente RECUBRIMIENTO DE ARISTAS a CIRCUITO HAMILTONIANO.*

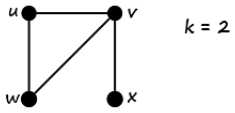
   Tomamos una instancia genérica de RECUBRIMIENTO DE ARISTAS, $G = (V, X)$ de n vértices y $k \in \mathbb{N}$ y la transformamos a una instancia de CIRCUITO HAMILTONIANO, $G' = (V', X')$, de forma tal que G tenga un recubrimiento de aristas de k o menos vértices si, y sólo si, G' es hamiltoniano.

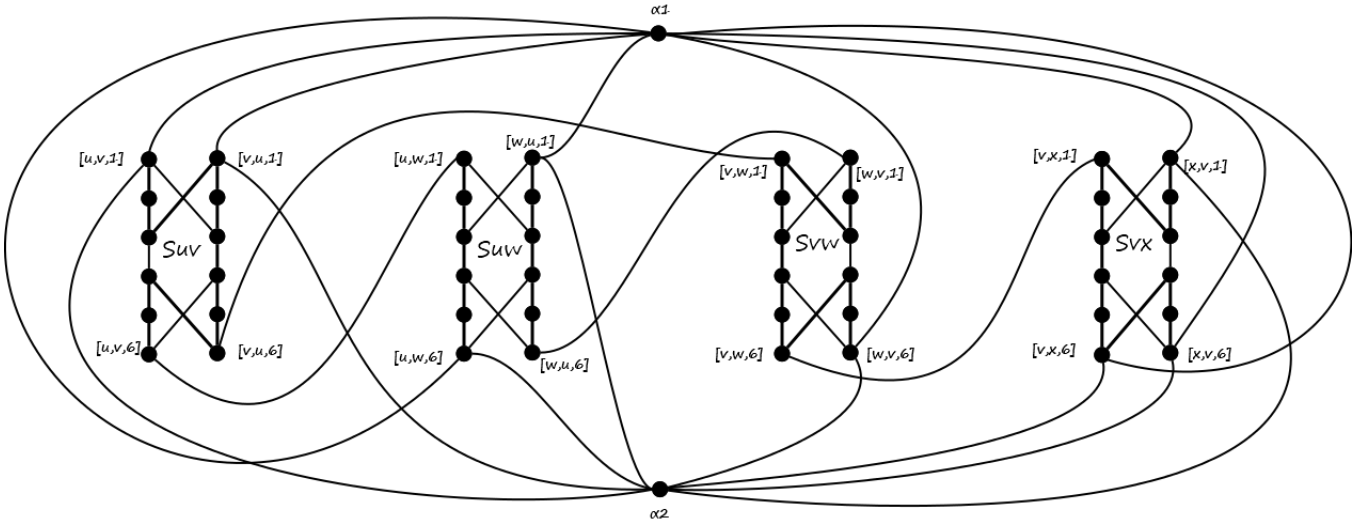
   Por cada arista $(u, w) \in X$, colocamos 12 vértices en G' con la siguiente estructura S_{uw}:



Dado un vértice $u \in V$, sean $w_1, \ldots, w_{d(u)}$ sus adyacentes. Entonces hacemos adyacentes en G' los vértices $[u, w_i, 6]$ y $[u, w_{i+1}, 1]$ para $i = 1, \ldots, d(u) - 1$.

Además, colocamos en V' k nuevos vértices $\alpha_1, \ldots, \alpha_k$ y los hacemos adyacentes a $[u, w_1, 1]$ y $[u, w_{d(u)}, 6]$.

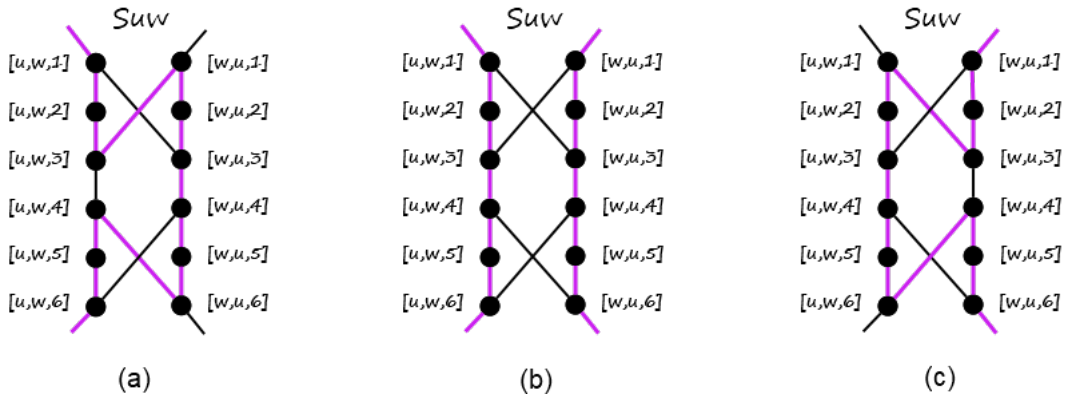




Esta transformación es polinomial. Ahora veamos que G tiene un recubrimiento de aristas de tamaño menor o igual a k si, y sólo si, G' tiene un circuito hamiltoniano.

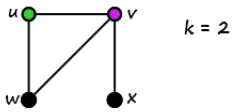
Supongamos que G tiene un recubrimiento de aristas, R, de tamaño k (si fuese de tamaño menor agregamos vértices para que sea de tamaño k), $R = \{r_1, \ldots, r_k\}$. Queremos ver que G' tiene un circuito hamiltoniano.

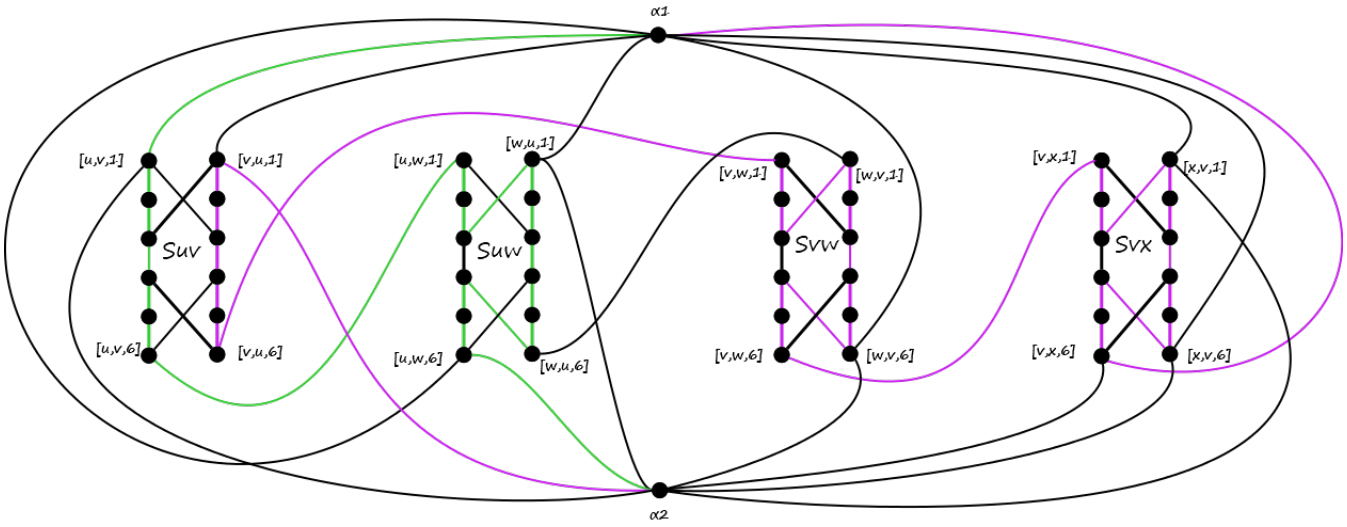
Como solamente los vértices con tercera coordenada 1 ó 6 tienen adyacentes fuera de la estructura y un circuito hamiltoniano debe recorrer todos los vértices, en cada estructura un circuito hamiltoniano debe seguir alguno de estos tres patrones:



Si el vértice $u \in R$ y $w \notin R$, vamos a elegir el patrón (a). En cambio, si $u \notin R$ y $w \in R$, nos quedaremos con el patrón (c). Si ambos vértices están en R, seguiremos el patrón (c).

Entonces armaremos el circuito hamiltoniano: $\alpha_1[r_1, w_{r_1}, 1] \dots [r_1, w_{r_1}, 6][r_1, w_{r_2}, 1] \dots [r_1, w_{d(r_1)}, 6]\alpha_2 \dots \alpha_1$.





Ahora, si G' tiene un circuito hamiltoniano H, debemos probar que G tiene un recubrimiento de aristas R de cardinal menor o igual a k. Como H recorre todos los vértices, pasará por todas las estructuras S siguiendo alguno de los tres patrones posibles. Si quitamos los vértices α_i del circuito hamiltoniano H, éste quedará partido en k subcaminos. Cada uno de estos subcaminos comenzará en un vértice de tercera coordenada 1, $[u, v, 1]$, atravesará la estructura S_{uv} siguiendo alguno de los tres patrones posibles, y saldrá de la estructura por el vértice $[u, v, 6]$, para luego finalizar o entrar en otra estructura por un vértice $[u, w, 1]$ y así continuar. Por el diseño de G', todas las estructuras que recorre un subcamino tienen un vértice en común, u, entrando y

saliendo de ellas por vértices de primera coordenada u. Ese vértice en común u formará parte del recubrimiento de aristas R. Como H recorre todos las estructuras (por recorrer todos los vértices), todas las aristas quedarán recubiertas por al menos un vértice de R, y como hay k subcaminos, el cardinal de R será k.

Ejemplo 18. *TSP es* **NP-completo**.

1. *En el Ejemplo 11 vimos que TSP es* **NP**.

2. *Para probar que es* **NP-completo**, *vamos a reducir polinomialmente CIRCUITO HAMILTONIANO a TSP.*

   Dado un grafo G vamos a construir un grafo completo G' con peso en las aristas y definir $k \in \mathbb{Z}_{\geq 0}$, de forma tal que G sea hamiltoniano si, y sólo si, G' tiene un circuito hamiltoniano de peso menor o igual a k.

   Si G tiene n vértices, G' será el grafo completo de n vértices. Fijaremos peso 0 para las aristas de G' que están en G y 1 para las que no están y definiremos $k = 0$. Entonces, claramente, G' tiene un circuito hamiltoniano de peso 0 si, y sólo si, G es hamiltoniano.

Muchos investigadores han intentado encontrar un algoritmo polinomial para resolver algún problema $\Pi \in$ **NP-completo**. Veamos por qué esto es tan importante.

Lema 2. *Si existe un problema* Π *en* **NP-completo** $\cap$ **P**, *entonces* **P**=**NP**.

Demostración. Si $\Pi \in$ **NP-completo** $\cap$ **P**, existe un algoritmo polinomial que resuelve Π, por estar en **P**. Por otro lado, como Π es **NP-completo**, para todo $\Pi' \in$ **NP**, $\Pi' \leq_p \Pi$.

Sea $\Pi' \in$ NP. Aplicando 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, se obtiene un algoritmo polinomial que resuelve Π'.

■

Hasta el momento no se conoce ningún problema en **NP-completo** $\cap$ **P**. Sin embargo, tampoco se ha demostrado que exista algún problema en **NP** $\setminus$ **P**. En ese caso se probaría que **P** $\neq$ **NP**.

7. La clase co-NP

Vimos que circuito hamiltoninano está en la clase **NP**. Pero consideremos ahora su versión *inversa*, ***circuito hamiltoniano complemento***:

Dado un grafo G, ¿es G no hamiltoniano?

¿Estará este problema también en **NP**? No sabemos la respuesta. Hasta el momento, la forma de verificar que un grafo general no tiene un circuito hamiltoniano es listar todas las permutaciones de sus vértices y verificar que ninguna define un cirtuito. Este certificado obviamente no es polinomial, por lo tanto no nos sirve para responder la pregunta.

Definición 15. El ***problema complemento*** de un problema de decisión Π, Π^c, es el problema de decisión cuyo conjunto de instancias es igual al de Π y responde ***SI*** para las instancias que Π responde ***NO*** y viceversa. Es decir Π^c es el problema de decisión tal que:

$$D_{\Pi^c} = D_\Pi \text{ y } Y_{\Pi^c} = D_\Pi \setminus Y_\Pi$$

Ejemplo 19. *El problema de primalidad y el problema de número compuesto son problemas complementarios.*

Ejemplo 20. *El problema complemento de TSP es:*

Dado un grafo $G = (V, X)$ con peso en sus aristas y $k \in \mathbb{N}$, ¿es verdad que todos los circuitos hamiltonianos de G tienen peso mayor que k?

Ejemplo 21. *Los problemas:*

- *Dado un grafo $G = (V, X)$, ¿es conexo?*
- *Dado un grafo $G = (V, X)$, ¿es disconexo?*

son problemas complemenatarios.

Es fácil ver el siguiente resultado:

Proposición 2. *Si un problema Π pertenece a* **P**, *entonces Π^c también pertenece a* **P**.

Demostración.
Como $\Pi \in$ **P**, existe un algoritmo polinomial para resolver Π. Este mismo algoritmo sirve para resolver Π^c invirtiendo la respuesta. ■

Este argumento no aplica para la clase **NP**. Es decir, si un problema Π está en **NP** no sirve este argumento para demostrar que Π^c está en **NP** (es más, no se sabe si ésto es cierto o no). Este concepto da origen a la siguiente definición:

Definición 16. Un problema de decisión pertenece a la clase **Co-NP** si dada una instancia de ***NO*** y evidencia polinomial de la misma, puede ser verificada en tiempo polinomial.

Ejemplo 22. *Circuito hamiltoniano complemento pertenece a la clase* **Co-NP**, *porque una instancia de* ***NO*** *es un grafo que tiene un ciruito hamiltoniano, y por lo tanto, podemos dar evidencia chequeable polinomialmente.*

La relación que podemos plantear entre estas clases es:

Proposición 3. *Si un problema Π pertenece a* **NP**, *entonces Π^c pertenece a* **Co-NP**.

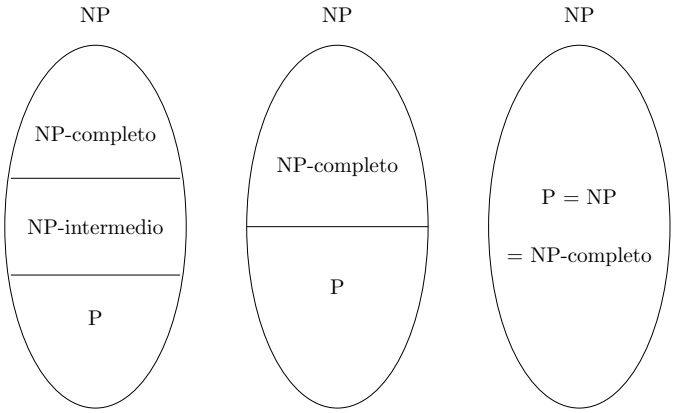
Demostración. Como $\Pi \in$ **NP**, para toda instancia de ***SI*** se puede dar evidencia de esto chequeable polinomialmente. Estas instancias son las instancias de ***NO*** de Π^c, y entonces esta misma evidencia sirve para chequear que son instancias de ***NO***. ■

8. Problemas abiertos de Teoría de Complejidad

Con estas nuevas definiciones tenemos los siguientes problemas abiertos:

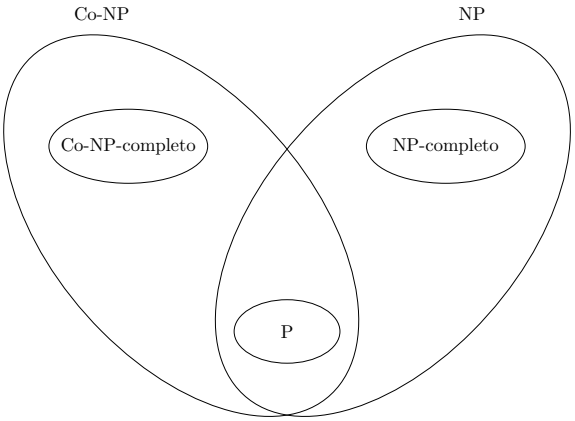
- ¿Es P = NP?
- ¿Es Co-NP = NP?
- ¿Es P = Co-NP ∩ NP?

La pregunta más relevante en el área es la primera, **¿Es P=NP?**. Éstos son tres mapas posibles para las clases de complejidad:



En la clase **NP-intermedio** están los problemas de **NP** que no son **P** ni **NP-completo**. Por supuesto no se sabe si hay algún problema en esta clase, ya que la exsistencia de este problema implicaría que **P** $\neq$ **NP**.

Ésta sería la situación si se probara que $P \neq NP$, $NP \neq Co-NP$, $P \neq Co-NP \cap NP$:



9. Extensión de un problema

Definición 17. El problema Π es una ***restricción*** de un problema $\bar{\Pi}$ si el dominio de Π está incluído en el de $\bar{\Pi}$.

Si Π es una restricción de $\bar{\Pi}$, se dice que $\bar{\Pi}$ es una ***extensión*** o ***generalización*** de Π.

Es intuitivo pensar que cuanto más general es el problema, más difícil es de resolver. Algunas veces, casos particulares (restricciones) de un problema **NP-completo** puede ser **P**. Pero no se puede dar la situación recíproca (ser el caso general más fácil que el caso particular).

Ejemplo 23.

- *3-SAT es una restricción de SAT. Ambos son problemas NP-completos.*
- *2-SAT es una restricción de SAT. 2-SAT es polinomial, mientras que SAT es NP-completo.*
- *COLOREO de grafos bipartitos es una restricción de COLOREO. Colorear un grafo bipartito es un problema polinomial, mientras COLOREO es NP-completo.*
- *CLIQUE de grafos planares es una restricción de CLIQUE. Encontrar una clique máxima de un grafo planar es un problema polinomial (porque sabemos que no puede tener a K_5 como subgrafo), mientras CLIQUE es NP-completo.*
- *CARTERO CHINO en grafos planares es una restricción de CARTERO CHINO. Ambos son problemas NP-completos [1].*

Formalmente podemos deducir que:

- Si $\bar{\Pi} \in \mathbf{P}$, entonces $\Pi \in \mathbf{P}$.
- Si $\bar{\Pi} \in \mathbf{NP}$, entonces $\Pi \in \mathbf{NP}$.
- Si $\Pi \in$ NP-Completo, entonces $\bar{\Pi} \in$ NP-Difícil.

10. Algoritmos Pseudopolinomiales

Definición 18. Un algoritmo para resolver un problema Π es ***pseudopolinomial*** si la complejidad del mismo es polinomial en función del valor de la entrada.

Ejemplo 24. *El problema de la mochila es NP-Completo, sin embargo, existe un algoritmo de complejidad $\mathcal{O}(nB)$ que lo resuelve, donde n es la cantidad de objetos y B el peso máximo que se puede cargar en la mochila.*

Una instancia de este problema es $c_1, \ldots, c_n, B$, por lo tanto su tamaño es $\mathcal{O}(nlogB)$. Por lo tanto $\mathcal{O}(nB)$ no es polinomial en el tamaño de la instancia, pero sí en el valor de la instancia.

Desde un punto de vista práctico, muchas veces los algoritmos pseudopolinomiales, a pesar de ser exponenciales, son aplicables.

11. Bibliografía recomendada

- M. Garey and D. Johnson, *Computers and intractability: a guide to the theory of NP- Completeness*, W. Freeman and Co., 1979.
- Capítulos 15 y 16 de C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Dover Publications INC, 1998.

Referencias

[1] C. H. Papadimitriou. On the complexity of edge traversing. *J. ACM*, 23(3):544–554, July 1976.