TEMA 4. NP completitud.

1- El concepto de reducción. Reducción polinómica.

En la clase NP tenemos una subclase (NPC) que contiene a los problemas más difíciles, la subclase de los problemas NP-completos. Para conocer mejor esta subclase tenemos que introducir primero las reducciones polinomiales.

Reducción polinomial de problemas:

Una reducción polinomial del lenguaje L1 al lenguaje L2 (**L1 αP L2**) es una reducción de L1 a L2 computable en tiempo determinístico polinomial. (Karp-reducción). Mf identificará la MT M que computa la función de reducción f (en este caso fϵFP (polinomiales)).

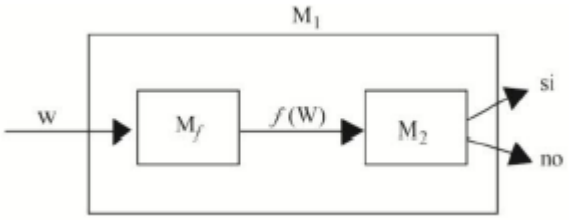
Si tenemos L1 α L2 implica que L2 es tan o más difícil que L1, desde el punto de vista de la computabilidad.

Con las reducciones polinomiales se puede establecer la misma relación, pero desde el punto de vista de la complejidad temporal:

Resultado1:

Sean L1, L2 lenguajes de P, siendo L2 ≠ Σ* y L2 ≠ Ø y con L1 αP L2: existe una MT que, a partir de una entrada w perteneciente (no perteneciente) a L1, lo que puede determinar en tiempo polinomial, genera una salida f(w) perteneciente (no perteneciente) a L2, lo que puede efectuar en tiempo constante, escribiendo un elemento que está (no está) en L2.

Teorema: L2 está en P(NP) y L1 αP L2 → L1 está en P (NP)



Corolario: Sean L1 y L2.

L1 ∉P (NP) y L1 αP L2 → L2 tampoco está en P (NP).

Podemos usar las reducciones polinomiales para probar que un problema no está en la clase P o en la clase NP.

La relación αP es reflexiva y transitiva. No es simétrica.

Sabemos que para todo lenguaje recursivo L, se cumple que L α LU pero no se cumple LU α L. Con αP ocurre lo mismo.

Mediante las reducciones polinomiales podremos definir una jerarquía temporal dentro de NP.

2- Problemas NP-difíciles y NP-completos.

Problemas NP-difíciles:

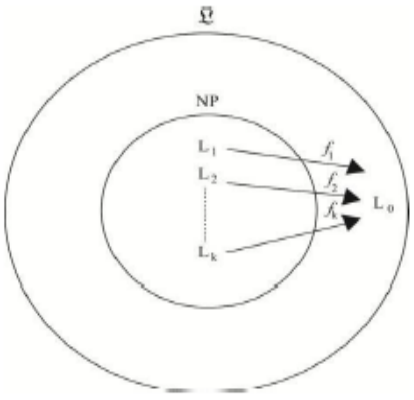
Un lenguaje L0 es NP-difícil (NP-hard, L0ϵNPH) ↔ para todo lenguaje LϵNP se cumple que L αP L0.

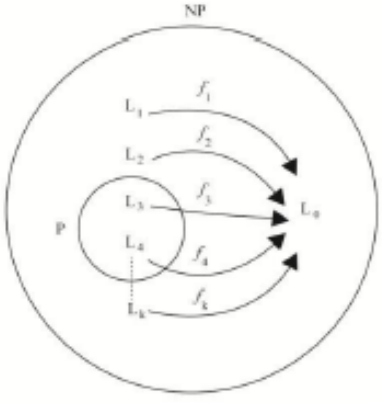
Un lenguaje L0 es NP-difícil ↔ todos los lenguajes de NP se reducen polinomialmente a él, no importa a qué clase pertenezca.

Problemas NP-completos:

Si L0ϵNP, entonces se dice que es NP-completo (L0ϵNPC).

Usaremos de forma indistinta lenguaje y problema NPH, NPC.



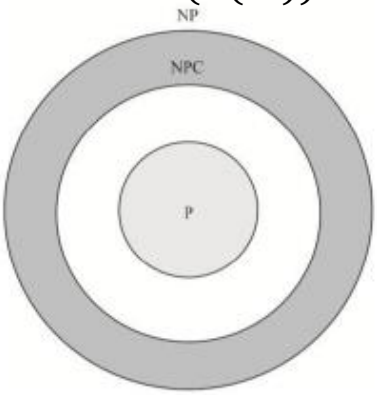


Con el siguiente teorema se formaliza que los problemas NP-completos son los más difíciles de NP.

Teorema: Si un problema NP-completo está en P → P = NP

Sea P ≠ NP. Probar que un problema es NPC equivale a que esté en NP-P.

Demostrando que un problema NPC se resuelve en tiempo determinístico T(n), tendríamos que NP ⊆ $\mathrm{DTIME}(\mathrm{T}(\mathrm{n}^{\mathrm{k}}))$.



Los problemas de NPC están en la franja de NP "más alejada" de P. Entre NPC y P queda una franja de problema que veremos más adelante.

Cuando tengamos un NPC, tendremos distintas alternativas para tratarlo: analizar caso promedio, usar algoritmos de aproximación, probabilísticos...

3- Algunos problemas NP-completos.

SAT: {determinar si una formula booleana es satisfacible}

Teorema: L1 ϵ NPC, L1 αP L2 y L2 ϵ NP → L2 ϵ NPC.

Quiere decir que encontrada una reducción polinomial de un lenguaje L1 NPC a un lenguaje L2 de NP, se prueba que L2 es NPC.

Reduciendo polinomialmente desde SAT encontraremos más lenguajes NPC.

CSAT:{reconocimiento de las formulas proposicionales satisfacibles que están en forma normal conjuntiva}

3-SAT: {determinar si una formula booleana en forma normal conjuntiva con 3 literales por clausula es satisfacible}

Problema del recubrimiento de vértices:{todas las aristas tocan a algún vértice del recubrimiento}

Problema del clique:{conjunto de vértices dos a dos adyacentes}

4- Algunas ideas sobre NP y NPC.

Todo lenguaje NPC se reduce polinomialmente a otro. Dada una función de reducción f entre dos lenguajes NPC L1 y L2, no necesariamente f^{-1} debería de ser una función de reducción de L2 a L1. Pero sí que sabemos que dados dos lenguajes L1 y L2 NPC, existe una función de reducción h de L1 a L2, tal que h^{-1} que es una función de reducción de L2 a L1. En este caso L1 y L2 son polinomialmente isomorfos (**p-isomorfos**).

Dada una reducción polinomial f entre los lenguajes L1 y L2 NPC, y una reducción polinomial g entre L2 y L1, se puede construir a partir de ellas una biyección h entre L1 y L2, tal que h y h^{-1} se computan en tiempo determinístico polinomial.

Conjetura de Berman-Hartmanis: establece que todos los lenguajes NPC son p-isomorfos. Si se demostrase, se tendría que P≠NP.

Nota: en NPC hay lenguajes finitos e infinitos.

Contra-Conjetura de Yoseph y Young: establece que pueden existir lenguajes NPC no p-isomorfos a SAT.

Los lenguajes **NPC son (exponencialmente) densos** → la cantidad de sus cadenas de longitud n no se pueden acotar por un polinomio p(n). Los lenguajes que sí cumplen esta propiedad se llaman polinomialmente densos o dispersos.

Si dos lenguajes L1 y L2 son p-isomorfos por medio de una función h que se computa en tiempo determinístico polinomial p(n), entonces sus densidades, dens1(n) y dens2(n), se relacionan polinomialmente:

- Las cadenas de L1 de longitud a lo sumo n se transforman mediante h en cadenas de L2 de longitud a lo sumo p(n).
  Como h es inyectiva, entonces dens1(n) ≤ dens2(p(n)).
- Si h^{-1} se computa en tiempo determinístico polinomial q(n), entonces por la misma razón dens2(n) ≤ dens1(q(n)).

→Ningún lenguaje NPC puede ser p-isomorfo a un lenguaje disperso.
La existencia de un lenguaje NPC disperso implica P=NP (los lenguajes más difíciles de NP no pueden tener “pocas” cadenas)