Contenido

Lista de Figuras

ANEXO A

TEORÍA DE LA COMPLEJIDAD

La teoría de la Complejidad Computacional es la parte de la teoría de la computación que estudia los recursos requeridos durante el cálculo para resolver un problema. Un cálculo resulta complejo si es difícil de realizar. En este contexto se puede definir la complejidad de cálculo como la cantidad de recursos necesarios para efectuar un cálculo. Así un cálculo difícil requerirá más recursos que uno de menor dificultad. Los recursos más estudiados son el tiempo (número de pasos de ejecución de un algoritmo para resolver un problema) y la memoria ocupada. Un algoritmo que resuelve un problema pero que se tarda mucho en hacerlo o consuma demasiada memoria difícilmente será de utilidad. A estos recursos se les puede añadir otros tales como el número de procesadores necesarios para resolver el problema en paralelo [COR04].

Así pues la teoría de la complejidad clasifica a los problemas de acuerdo a la dificultad para resolverlos.

1. Criterios para clasificación de problemas.

A continuación se observan las clases de problemas, pero antes se introducen algunos conceptos claves para dicha clasificación.

1.1. **Problema abstracto:** un problema abstracto *Q* es una relación binaria sobre un conjunto *I* de instancias de problemas y un conjunto *S* de soluciones al problema.

1.2. **Problemas de decisión**: aquellos que tienen una solución de tipo si/no. En este caso se puede ver un problema de decisión abstracto como una función que mapea la instancia del conjunto I al conjunto solución {0,1}.

1.3. **Problemas de optimización:** A diferencia de los problemas de decisión, los problemas de optimización buscan que algunos valores sean minimizados o maximizados.

1.4. **Codificaciones:** una codificación es una manera de representar un número de expresiones en términos de otras (usualmente más simples). Sin embargo, se pueden codificar múltiples expresiones como una sola expresión [WER07]. Dicho de otra manera, una codificación "*e"* mapea un conjunto *S* en cadenas binarias. Las codificaciones se utilizan para mapear problemas abstractos a problemas concretos.

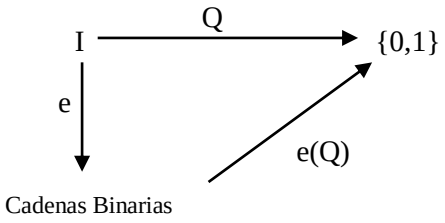

Figura A.1. Codificaciones. Figura tomada de [SDS95]

1.5. **Problema concreto:** un problema concreto es una relación binaria sobre un conjunto de instancias de problemas *I* y un conjunto de soluciones *S*.

Se dice que un algoritmo resuelve un problema concreto en un tiempo $O(T(n))$ si cuando se le pasa una instancia i de tamaño $n = |i|$, el algoritmo puede producir una solución en al menos un tiempo $O(T(n))$.

Un problema concreto se puede resolver en un *tiempo polinomial* si existe un algoritmo que lo resuelva en un tiempo $O(n^k)$ para alguna constante *"k"*.

1.6. **Algoritmos de verificación:** dado un algoritmo A de dos argumentos, donde un argumento es una cadena ordinaria *"x"* de entrada y el otro es una cadena binaria *"y"* denominada certificado. Un algoritmo A verifica una cadena de entrada de *"x"* si existe un certificado *"y"* tal que $A(x, y) = 1$. El lenguaje verificado por un algoritmo de verificación A es:

$$L = \{x \in \{0,1\}^* : donde\ existe\ y \in \{0,1\}^*\ tal\ que\ A(x, y) = 1\ \}$$

Intuitivamente un algoritmo A verifica un lenguaje L si para toda cadena $x \in L$ existe un certificado *"y"* que *A* puede utilizar para probar que $x \in L$.

1.7. **Reducciones:** un problema *Q* puede reducir a otro problema *Q'* si cualquier instancia de *Q* se puede replantear fácilmente como instancia de *Q'*, cuya solución permite encontrar un solución para la instancia de *Q*. Así si un problema *Q* reduce a otro problema *Q'* entonces *Q* no es, en un sentido, más difícil de resolver que *Q'*.

Entonces un lenguaje L_1 es reducible en tiempo polinomial a un lenguaje L_2, escrito $L_1 \leq_P L_2$, si existe una función computable en tiempo polinomial:

$$f: \{0,1\}^* \to \{0,1\}^*, x \in L_1 \; si \; y \; solo \; si \; f(x) \in L_2.$$

En el cual *"f"* se conoce como la *función de reducción* y un algoritmo *F* de tiempo polinomial que compute *"f"* se denomina *algoritmo de reducción.*

2. Clasificación de Problemas.

Ahora que ya se tienen algunos conceptos importantes introducidos se puede pasar a definir las clases de problemas de acuerdo a la teoría de la complejidad.

2.1. Clase de complejidad P:

Se puede decir que los problemas de la clase P son ***tratables*** en el sentido de que suelen ser abordables en la práctica. La clase de problemas P corresponde al conjunto de problemas de decisión concretos que se pueden resolver en tiempo polinomial.

2.2. Clase de complejidad NP:

El nombre NP significa "Tiempo Polinomial No determinista", son considerados problemas ***intratables*** y se distinguen por el hecho de que se puede aplicar un algoritmo de tiempo polinomial para comprobar si una posible solución es válida o no. Esto se logra aplicando heurísticas para obtener soluciones hipotéticas que se van desestimando o aceptando en un tiempo polinomial.

Esta clase de complejidad NP es la clase de lenguajes que pueden verificarse por un algoritmo de tiempo polinomial. Más precisamente un lenguaje *L* pertenece a *NP* si y solo si existen un algoritmo *A* de tiempo polinomial, dos entradas y una constante *"c"* tal que:

$$L = \left\{ \begin{matrix} x \in \{0,1\}^* : existe \; un \; certificado \; "y" \; con \; |y| = O(|x|^c) \\ tal \; que \; A(x,y) = 1 \end{matrix} \right\}$$

Se dice entonces que un algoritmo *A* verifica al lenguaje *L* en un tiempo polinomial.

Hay que tener en cuenta que si $L \in P$ entonces $L \in NP$, puesto que si existe un algoritmo de tiempo polinomial que decida *L*, se puede convertir fácilmente a un algoritmo de verificación con dos argumentos simplemente ignorando cualquier certificado y aceptando exactamente esas cadenas de entrada y determinar que esta en *L*. Así pues $P \subseteq NP$.

En realidad se desconoce si P=NP, pero la mayoría de los investigadores creen que P y NP no son de la misma clase. Intuitivamente la clase P consiste de los problemas que se pueden resolver fácilmente y la clase NP de los problemas para los cuales una solución se puede verificar fácilmente. Está comprobado que a menudo es más difícil resolver un problema desde el principio que verificar una solución que esté presente, especialmente cuando se trabaja bajo restricciones de tiempo. Teóricamente un científico de computación generalmente cree que esta analogía se extiende para clases P y NP, y que NP incluye lenguajes que no están en P.

Hay una evidencia más convincente de que $P \neq NP$ y es justamente la existencia de los lenguajes NP-completos, de los cuales se hablará más tarde.

2.3. Clase de complejidad coNP:

Antes de empezar con la clase NP-completo se hablará de la clase coNP. A pesar del trabajo de muchos investigadores, nadie aún sabe si la clase NP está cerrada bajo un complemento. Es decir, ¿hacer $L \in NP$ implica que $\overline{L} \in NP$? Se pude definir entonces, la clase de complejidad coNP como el conjunto de lenguajes *L* tal que $\overline{L} \in NP$. La pregunta es si NP está cerrado bajo el complemento se puede ser reformular como $NP = coNP$ (ver Fig. A.2(a)). Puesto que P está cerrado bajo complemento, sigue que $P \subseteq \text{NP} \cap \text{coNP}$ (fig A.2 (b)). Una vez nuevamente, sin embargo, es desconocido si $P = NP \cap \text{coNP}$ (fig A.2 (c)) o si existe algún lenguaje en $\text{NP} \cap \text{coNP} - \text{P}$ (fig A.2 (d)). [CLR90] La figura A.2 muestra los cuatro posibles escenarios.

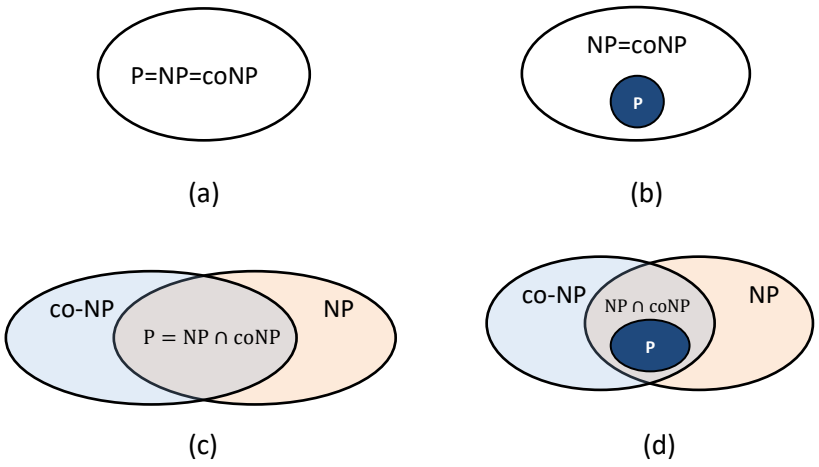

Figura A.2.Cuatro relaciones entre las clases de complejidad.

2.4. Clase de problemas NP completo (NPC)

Las reducciones polinomiales proveen un significado formal para mostrar que un problema es tan duro como otro, por lo menos con un factor constante. Esto es, si $L_1 \leq_p L_2$, entonces L_1 no es más que un factor polinomial más duro que L_2. Un lenguaje L es NP-completo si:

- $L \in NP$
- $L' \leq_p L \ \forall \ L' \in NP$

Quizá la razón por la cual los científicos en computación teórica creen que $P \neq NP$ es la existencia de la clase de problemas NP-completo. Esta clase tiene la sorprendente propiedad que si algún problema NP-completo se puede resolver en un tiempo polinomial entonces todos problemas en NP tienen una solución en tiempo polinomial, que es $P = NP$. A pesar de los años de estudio no se ha descubierto aún algún algoritmo de tiempo polinomial para algún problema NP-completo. Actualmente existe un premio de un millón de dólares entregado por el Clay Mathematics Institute para el que consiga semejante solución [CLA07].

Muchos científicos de la teoría computacional miran la relación entre P, NP y NP, así: P y NPC están completamente contenidos dentro de NP, y $\mathrm{P} \cap \mathrm{NPC} = \emptyset$ ver figura A.3.

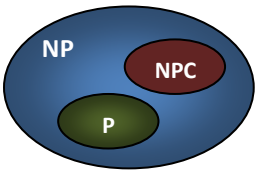

Figura A.3. Posible relación entre las clases P, NP y NPC.

Un ejemplo de problemas NPC es el famoso problema del agente viajero (TSP por sus siglas en inglés), el cual se encuentra al final de este anexo.

Otro aspecto importante es que para poder aplicar la teoría de problemas NP-completos se debe replantear los problemas de optimización como problemas de decisión. Típicamente un problema de optimización se puede replantear imponiendo un límite en el valor a ser optimizado.

Aunque la teoría de problemas NP-completo obliga a replantear un problema de optimización como un problema de decisión, este requerimiento no disminuye el impacto de la teoría. En general si se puede resolver un problema de optimización fácilmente, se pude además resolver su problema de decisión relacionado fácilmente comparando el valor obtenido del problema de optimización con un valor límite entregado como entrada al

problema de decisión. Si un problema de optimización es fácil entonces su problema de decisión relacionado es también fácil. Además si un problema de decisión es difícil también podemos decir que su problema relacionado de optimización es difícil.

2.5. Clase NP duros

Corresponde a los problemas de decisión que son por lo menos tan difíciles como un problema de NP. Si un lenguaje *L* satisface la condición 2 de la definición de NP-completo y no necesariamente la condición 1 se dice que es NP-Duro. *Ya* que si se encuentra un algoritmo *A* que resuelve uno de los problemas *H* de NP-duro en tiempo polinomial, entonces es posible encontrar algoritmos que trabajen en tiempo polinomial para cualquier problema de la clase NP ejecutando primero la reducción de este problema en *H* y luego ejecutando el algoritmo *A*. La clase NP-completo puede definirse entonces como la intersección entre NP y NP-duro [UNI07] ver figura A.4.

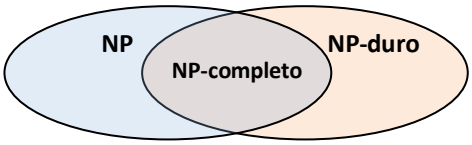

Figura A.4. La clase NP-duro

2.6. Problema del Agente Viajero (TSP).

Se enuncia como sigue: sean *N* ciudades de un territorio. El objetivo es encontrar una ruta que comenzando y terminando en una ciudad concreta pase una sola vez por cada una de las ciudades y minimice la distancia recorrida por el viajero. Es decir, encontrar una permutación $P = \{c0, c2, \ldots, cn - 1\}$ tal que $d_p = \sum_{i=0}^{N-1} d[c_i, c_i + 1 mod(N)]$sea mínimo. La distancia entre cada ciudad viene dada por la matriz *D*: NxN, donde *d[x, y]* representa la distancia que hay entre la ciudad *X* y la ciudad *Y*.

Así pues se encuentra que el número posible de combinaciones viene dado por la factorial del número de ciudades $N!$, esto hace que la búsqueda de la solución sea intratable incluso para valores moderados de *N*.

Referencias

[COR04] Cortéz. A. *Teoría de la complejidad computacional y Teoría de la Computabilidad*. Rev. Investig. Sist. Inform. Facultad de Ingeniería de Sistemas e Informática. Universidad Nacional Mayor de San Marcos. ISSN: 1815-0268. Perú 2004.

[WER07] Weisstein, Eric W. "Encoding." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Encoding.html

[SDS95] Combinatorial Algorithms .San Diego State University . 1995 CS 660: http://www.eli.sdsu.edu/courses/fall95/cs660/notes/NP2/NP2.html

[CLR90] Cormen. T, Leiserson. C, Rivest.R. Introduction to Algorithms, MIT Press. ISBN 0-262-03141-8. 1990.

[CLA07] Millennium Problems. Clay Mathematics Institute. http://www.claymath.org/millennium/

[UNI07] Explicación de Problemas. Algorítmos Genéticos http://lear.inforg.uniovi.es/ia/Genetico-TSP/Espacio_busqueda.htm