Fundamentos de Programación Entera
4. Complejidad computacional

Carlos Testuri – Fernando Islas

Departamento de Investigación Operativa – Instituto de Computación
Facultad de Ingeniería – Universidad de la República

2012-2023

Contenido

Problema computacional

Problema, instancia, y algoritmo y tiempo de ejecución

Se denomina *problema computacional* a una cuestión abstracta planteada a partir de entidades y resuelta mediante algoritmos. Sobre las entidades se establecen métricas de tamaño que permiten dimensionar el problema.

Una *instancia de un problema* se define a partir de datos que determinan las entidades del problema. Esta tiene determinado el *tamaño* de las entidades.

Un *algoritmo* de resolución de un problema se define como un conjunto de instrucciones que se aplica para resolver instancias del mismo. Se dice que un algoritmo resuelve un problema si, para toda instancia del mismo, produce una respuesta correcta mediante la ejecución de una cantidad finita de instrucciones.

El *tiempo de ejecución* de un algoritmo se define en función de la cantidad de instrucciones ejecutadas. Se puede asumir que cada instrucción emplea una unidad de tiempo. El análisis de complejidad será de base temporal.

Algoritmo eficiente

El tiempo de ejecución depende del tamaño de la instancia del problema.

Para un algoritmo, se define como $T(n)$ al *tiempo de ejecución del peor caso* de las instancias de tamaño n.

Se dice que un algoritmo se ejecuta en *tiempo polinomial* si existe un entero k tal que $T(n) =: O(n^k)$.

Se dice que un algoritmo se ejecuta en *tiempo pseudo-polinomial* si $T(n)$ es función polinomial de datos (valores numéricos) de las instancias.

Se dice que un *algoritmo es eficiente* si se ejecuta en tiempo polinomial.

Complejidad computacional

La teoría de complejidad computacional permite clasificar los problemas computacionales según su grado de dificultad de resolución.

Un objetivo es determinar cuando un problema puede ser resuelto mediante un uso de orden polinomial del recurso tiempo (cantidad de operaciones), en función de las dimensiones del problema (ej. tamaño entrada).

Para el problema general de programación entera y la mayoría de sus casos especiales no se conoce un algoritmo de resolución eficiente.

Desde 1960's *no se ha podido probar* que dichos algoritmos no existan.
Se conjetura que no existen y que los problemas son difíciles de resolver.

Problemas según dificultad de resolución

Resolubles eficientemente

Determ. de lotes no-cap.(ULS)
Flujo en red
Camino más corto
Flujo máximo
Transporte
Asignación
Arbol de exp. minimal (MST)
Programación entera con TU

Resolución eficiente desconocida

Mochila
Cobertura, empaque y partición
Vendedor viajero (TSP)
Localización inst. no-cap. (UFL)
Arbol de Steiner
Programación entera (IP)

¿Cómo caracterizar los problemas según dificultad de resolución?

Mecanismo de comparación: Reducción en tiempo polinomial

Para clasificar problemas según grado de dificultad se utiliza la comparación:

"El problema Y no es más difícil que el problema X".

Reducción. Dado Y que no es más difícil que X. Si se dispone de una "caja negra" para resolver X y un mecanismo de reducción/transformación de las instancias de Y en instancias de X entonces se puede resolver Y.

Reducción en tiempo polinomial. Si cualquier instancia del problema Y puede ser resuelta realizando una cantidad polinomial de pasos más *una cantidad polinomial de invocaciones* a la caja negra que resuelve el problema X, entonces se dice que

"Y es reducible en tiempo polinomial a X" y se denota $Y \leq_P X$.

Propiedades de la reducción en tiempo polinomial

Proposición (1)

*Sea $Y \leq_P X$. Si **X puede** resolverse en tiempo polinomial, entonces **Y puede** resolverse en tiempo polinomial.*

¿Cómo se podría demostrar?
¿Cómo se puede aplicar a los problemas Flujo en Red y Transporte?

Proposición (2)

*Sea $Y \leq_P X$. Si **Y no puede** resolverse en tiempo polinomial, entonces **X no puede** resolverse en tiempo polinomial.*

Se denomina problema difícil al que no puede resolverse eficientemente.
Si se tiene un problema Y que se conoce que es difícil y se demuestra que $Y \leq_P X$ entonces X debe ser difícil.
¿Qué relación hay entre (1) y (2)?

Problemas de decisión

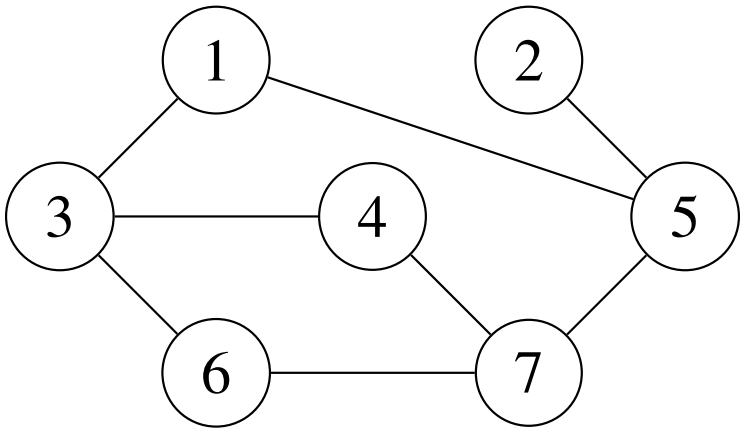
La teoría de complejidad es establecida sobre problemas de decisión.
Los *problemas de decisión* retornan las respuestas: Si o No.

Por lo que se necesita representar los problemas de optimización en términos de problemas de decisión asociados.

Dado el problema de optimización $\max\{c(x) : x \in X\}$, se puede establecer el problema de decisión equivalente en términos de dificultad de resolución:
- Dada una constante K, ¿existe $x \in X$ con valor $c(x) \geq K$?

Problema Conjunto Independiente

Versión optimización Dado un grafo $G = (V, E)$ se dice que un subconjunto de sus vértices $S \subseteq V$ es *independiente* si ningún par de vértices en S comparten arista. Se requiere encontrar el conjunto independiente más grande.
Dada la instancia

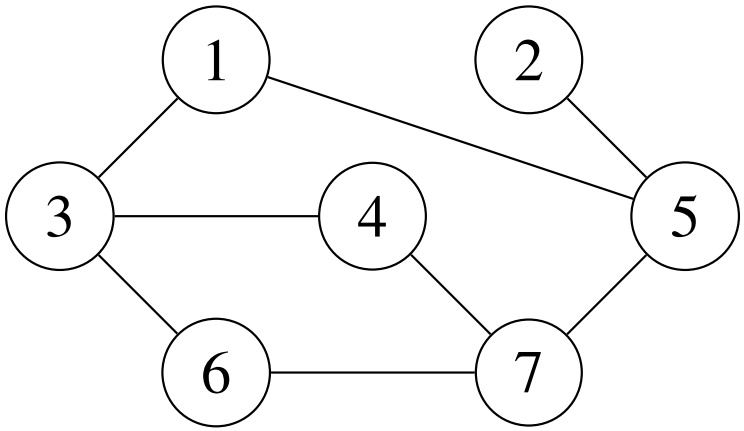


¿Cuál es su conjunto independiente más grande?
La metodología requiere problemas de decisión equivalentes en complejidad.
Versión decisión Dado un grafo G y un número k.
¿Contiene G un conjunto independiente de tamaño al menos k?

No se conoce como resolverlo en tiempo polinomial.

Problema Cobertura por Vértices

Versión optimización Dado un grafo $G = (V, E)$ se dice que un subconjunto de sus vértices $S \subseteq V$ es una *cobertura (de aristas) por vértices* si cada arista tiene al menos un vértice en S. Se requiere encontrar el conjunto de cobertura más pequeño.
Dada la instancia



¿Cuál es su cobertura por vértices más pequeña?
Versión decisión Dado un grafo G y un número k.
¿Contiene G una cobertura por vértices de tamaño a lo sumo k?

No se conoce como resolverlo en tiempo polinomial.

Equivalencia entre *conjunto independiente* y *cobertura por vértices*

Proposición (3)

Sea el grafo $G = (V, E)$. $S \subseteq V$ es un conjunto independiente si y solo si su complemento $V \setminus S$ es una cobertura por vértices.

Prueba.

$\Rightarrow$ Supongamos que S es un conjunto independiente. Sea una arista (v, w), dado que S es conjunto independiente, entonces v y w no pueden estar ambos en S. Por lo cual un de ellos debe estar en $V \setminus S$. Por lo cual cada arista tiene un vértice en $V \setminus S$. Por lo tanto $V \setminus S$ es una cobertura por vértices.
$\Leftarrow$ Supongamos que $V \setminus S$ es una cobertura por vértices. Sean los vértices v y w de S. Si v y w estuvieran adjuntos por una arista e, entonces ningún extremo de e estaría en $V \setminus S$, contradiciendo la suposición de que $V \setminus S$ es una cobertura por vértices. Por lo tanto, ningún par de vértices en S son adjuntos por una arista. Por lo tanto S es un conjunto independiente. □

Reducciones entre Conjunto Independiente y Cobertura por Vértices

Proposición (4)

Conjunto Independiente $\leq_P$ *Cobertura por Vértices.*

Prueba.

Según (3). Si se tiene una caja negra para resolver Cobertura por Vértices, entonces se puede decidir si G tiene una conjunto independiente de tamaño al menos k consultando a la caja negra si G tiene una cobertura por vértice de tamaño a lo sumo $n - k$. □

Proposición (5)

Cobertura por Vértices $\leq_P$ *Conjunto Independiente.*

¿Cómo podría demostrase?

Relación con Problemas de Cobertura y Empaque en Conjunto

Dados los conjuntos $M = \{1, ..., m\}$ y $N = \{1, ..., n\}$.
Sea M_i, con $i \in N$, una colección de subconjuntos de M.
Dado S subconjunto de N,

- S se dice *cobertura* de M si $\cup_{i \in S} M_i = M$,
- S se dice *empaque* de M si $M_i \cap M_j = \emptyset$, para todo $i, j \in S, i \neq j$,

En el Problema de *Cobertura* en M se busca S de *menor* tamaño.
En el Problema de *Empaque* en M se busca S de *mayor* tamaño.

Proposición (6)

Cobertura por Vértices $\leq_P$ *Cobertura en Conjunto.*

Proposición (7)

Conjunto Independiente $\leq_P$ *Empaque en Conjunto.*

¿Cómo podrían demostrase?

Reducción Cobertura por Vértices a Cobertura en Conjunto

Proposición (6)

Cobertura por Vértices $\leq_P$ *Cobertura en Conjunto.*

Prueba.

Se requiere transformar una instancia cualquiera (general) de Cobertura por Vértices a una instancia de Cobertura en Conjunto, invocar a la caja negra de Cobertura en Conjunto con la instancia y retornar la respuesta de la caja.
La instancia de Cobertura por Vértices esta definida por un grafo $G = (V, E)$ y un número k. Se requiere cubrir las aristas de E en forma similar a los elementos del conjunto M. Por cada vértice $i \in V$ y sus aristas incidentes en Cobertura por Vértices se agrega un conjunto M_i de estas aristas en Cobertura en Conjunto.
Se tiene que M puede ser cubierto por a lo sumo k de los conjuntos M_1, ..., M_n si y solo si G tiene una cobertura por vértices de tamaño a lo sumo k. Si M_{i_1}, ..., M_{i_j}, con $j \leq k$, es una cobertura de M, entonces cada arista en G es adyacente a uno de los vértices i_1, ..., i_j, y el conjunto i_1, ..., i_j es una cobertura por vértices en G de tamaño $j \leq k$, por lo tanto M_{i_1}, ..., M_{i_j} es una cobertura de M. Se invoca a la caja negra con la instancia generada y se responde Si si y solo si la caja negra responde Si. □

Problema de Satisfacibilidad Booleana (SAT)

Es un problema en álgebra Booleana. Sea un conjunto de variables Booleanas $X = \{x_1, x_2, ..., x_n\}$. Se denomina término, t, sobre X a la expresión de una variable x_i o su negación $\overline{x_i}$, $t \in \{x_1, x_2, ..., x_n, \overline{x_1}, \overline{x_2}, ..., \overline{x_n}\}$.
Se denomina cláusula a la disyunción de diferentes términos $C = t_1 \vee t_2 ... \vee t_\ell$.
Se busca satisfacer la conjunción de un conjunto de cláusulas diferentes, $C_1 \wedge C_2 ... \wedge C_k$, a partir de la asignación de valores a las variables.
Problema SAT. ¿Existe una asignación de las variables que satisface todas las cláusulas?

Existe un caso especial de SAT (minimal) igualmente difícil que el general.

Problema 3-SAT. Dado un conjunto de cláusulas, **cada una de 3 términos**, sobre un conjunto de variables. ¿Existe una asignación de las variables que satisface todas las cláusulas?

¿Cómo podría resolverse SAT?

Reducción de 3-SAT a Conjunto Independiente 1/3

Proposición (8)

3-SAT $\leq_P$ *Conjunto Independiente.*

Prueba.

Se cuenta con una caja negra para resolver Conjunto Independiente y se requiere resolver una instancia de 3-SAT con $X = \{x_1, ..., x_n\}$ y cláusulas $C_1, ..., C_k$. Una técnica de resolución de 3-SAT es elegir un término de cada cláusula y tratar de que sea verdadero. Para la selección se busca que cada par de términos no contengan uno de ellos una variable y el otro la negación de la variable.
Se construye un grafo $G = (V, E)$ con $3k$ vértices agrupados en k triángulos. Para cada $i = 1, ..., k$ se construyen los vértices v_{i1}, v_{i2}, v_{i3} conectados por aristas.
...

Reducción de 3-SAT a Conjunto Independiente 2/3

Prueba (cont.)

Solo se puede seleccionar un vértice de cada triángulo, lo que corresponde a seleccionar un término con valor 1 en cada cláusula. Para representar los conflictos entre las variables de los términos se agrega una arista entre las variables.
La instancia de 3-SAT se satisface si y solo si el grafo G tiene un conjunto independiente de tamaño al menos k.
Por una parte, si la instancia de 3-SAT se satisface, entonces cada triángulo contiene al menos un vértices cuya variable vale 1. Sea S un conjunto compuesto por los vértices cuyas variables valen 1. El conjunto S es independiente, dado que si hubiera una arista entre un par de vértices de S entonces sus variables estarían en conflicto, lo cual no es posible.
Por la contraria, se supone que G tiene un conjunto independiente S de tamaño al menos k. Entonces el tamaño de S es exactamente k y consiste de un vértice de cada triángulo.
...

Reducción de 3-SAT a Conjunto Independiente 3/3

Prueba (cont.)

Existe una asignación de las variables de la instancia de 3-SAT tal que corresponde con los vértices de S. Para cada variable x_i, si ni x_i ni $\overline{x_i}$ corresponde con un vértice en S entonces se asigna arbitrariamente $x_i = 1$.
De lo contrario, exactamente x_i o $\overline{x_i}$ corresponderían con un vértice en S, porque si un vértice en S estuviera etiquetado x_i y otro etiquetado $\overline{x_i}$, entonces habría una arista entre estos dos vértices, contradiciendo la suposición de que S es un conjunto independiente.
Por tanto, si x_i es una etiqueta de un vértice en S, se establece $x_i = 1$, y de lo contrario $x_i = 0$. Al construir la asignación de esta manera, todas las etiquetas de los vértices en S se evaluarán a 1. □

Transitividad de reducciones en tiempo polinomial

Proposición (9)

Si $Z \leq_P Y$ *y* $Y \leq_P X$ *entonces* $Z \leq_P X$

Prueba.

Dada una caja negra para X, se requiere resolver una instancia de Z. Se componen los dos algoritmos asociados a $Z \leq_P Y$ y $Y \leq_P X$. Se ejecuta el algoritmo para Z usando una caja negra para Y; cada vez que la caja negra para Y es invocada, se lo hace en una cantidad polinomial de pasos invocando el algoritmo que resuelve instancias de Y usando la caja negra para X. □

Ejemplo

$$\text{3-SAT} \leq_P \text{Conjunto Independiente} \leq_P \text{Cobertura por Vértices} \leq_P \text{Cobertura en Conjunto}$$

$$\Rightarrow$$

$$\text{3-SAT} \leq_P \text{Cobertura en Conjunto}$$

Obtener y comprobar soluciones de problemas

Además de comparar los problemas se quiere determinar un atributo distintivo de los problemas para clasificarlos.

Ejemplos. Dada una supuesta solución de una instancia

- del problema de decisión de Conjunto Independiente, o
- del problema 3-SAT.

¿Cómo se puede probar que es una solución de la instancia?

En ambos problemas es más difícil *obtener* una solución que *comprobar* una supuesta solución.

¿Cómo se podría medir el trabajo de comprobar una supuesta solución?
Para ambos problemas no se conoce un algoritmo eficiente para obtener soluciones, pero una supuesta solución se puede comprobar eficientemente.

Una instancia de 3-SAT es *satisfacible* cuando tiene al menos una solución.
¿Cómo probar que una instancia de 3-SAT *no es satisfacible*?

Problemas de decisión y algoritmos - Clase de problemas $\mathcal{P}$

Una entrada (instancia) de un problema de decisión se codifica como una cadena binaria finita s, cuyo largo se denota por $|s|$.
Un problema de decisión X se identifica con el conjunto de entradas en las que la respuesta es Si.
Un algoritmo A para un problema de decisión X recibe una entrada s y retorna la salida $A(s) \in \{\text{Si}, \text{No}\}$.
Se dice que A resuelve el problema X si para toda cadena s se cumple $A(s) = \text{Si}$ si y solo si $s \in X$.
Se dice que el algoritmo A tiene tiempo de ejecución polinomial si existe una función polinomial $f(\cdot)$ tal que para cada cadena de entrada s, el algoritmo termina en $O(f(|s|))$ pasos.

Se denomina *clase de problemas* $\mathcal{P}$ al conjunto de problemas para los que existen algoritmos que los resuelven en tiempo polinomial.

Certificador eficiente

Formaliza la comprobación eficiente de una solución de un problema.
Para verificar una solución de una entrada s de un problema se utiliza una cadena t, denominada "certificado", que contiene la prueba de que s es una instancia Si del problema.
Se dice que un algoritmo B es un certificador eficiente para un problema X si se cumplen:

- B es un algoritmo de tiempo polinomial, con argumentos s y t.
- Existe una función polinomial p tal que para cada cadena s, se cumple que $s \in X$ si y solo si existe una cadena t tal que $|t| \leq p(|s|)$ y $B(s,t) = \text{Si}$.

Ejemplos según problemas:

- 3-SAT: el certificado t es la asignación de los valores de las variables, el certificador B evalúa la conjunción de las cláusulas para dicha asignación.
- Conjunto Independiente: el certificado t es el conjunto solución de al menos k vértices; el certificador B comprueba, para estos vértices, que ninguna arista conecta cualquier par de ellos.

Clase de problemas $\mathcal{NP}$

Se define la clase de problemas $\mathcal{NP}$ como el conjunto de todos los problemas para los que existe un certificador eficiente.

Proposición (10)

$\mathcal{P} \subseteq \mathcal{NP}$

Prueba.

Dado un problema $X \in \mathcal{P}$, existe un algoritmo A de tiempo polinomial que resuelve X. Demostrar que $X \in \mathcal{NP}$ implica demostrar que existe un certificador eficiente B para X. B se puede construir a partir de A, dado que A es de tiempo polinomial. □

El algoritmo certificador, B, podría ser usado para resolver el problema X, buscando exhaustivamente en el conjunto de todos los posibles certificados t, cuyo tamaño es potencialmente infinito. Dicho proceso de búsqueda se equipara con una máquina de Turing no-determinista (Teoría de Autómatas). El nombre $\mathcal{NP}$ proviene de "tiempo polinomial no-determinista".

¿Existe un problema en $\mathcal{NP}$ que no pertenece a $\mathcal{P}$?

No se ha podido probar que los problemas, que no han podido resolverse eficientemente, requieran tiempo supra-polinomial para resolverse.

No se ha podido probar que exista un problema en $\mathcal{NP}$ que no pertenezca a $\mathcal{P}$.

¿Es $\mathcal{P} = \mathcal{NP}$?

A partir del esfuerzo realizado desde 1960's se conjetura que $\mathcal{P} \neq \mathcal{NP}$.

Clase de problemas $\mathcal{NP}$-completo

Formalización de la clase de los problemas más difíciles en $\mathcal{NP}$.
Para lo cual se utiliza el mecanismo de *reducción en tiempo polinomial*.

Se dice que un problema X es difícil o que $X \in \mathcal{NP}$-completo si:

i. $X \in \mathcal{NP}$,

ii. para todo $Y \in \mathcal{NP}$, $Y \leq_P X$.

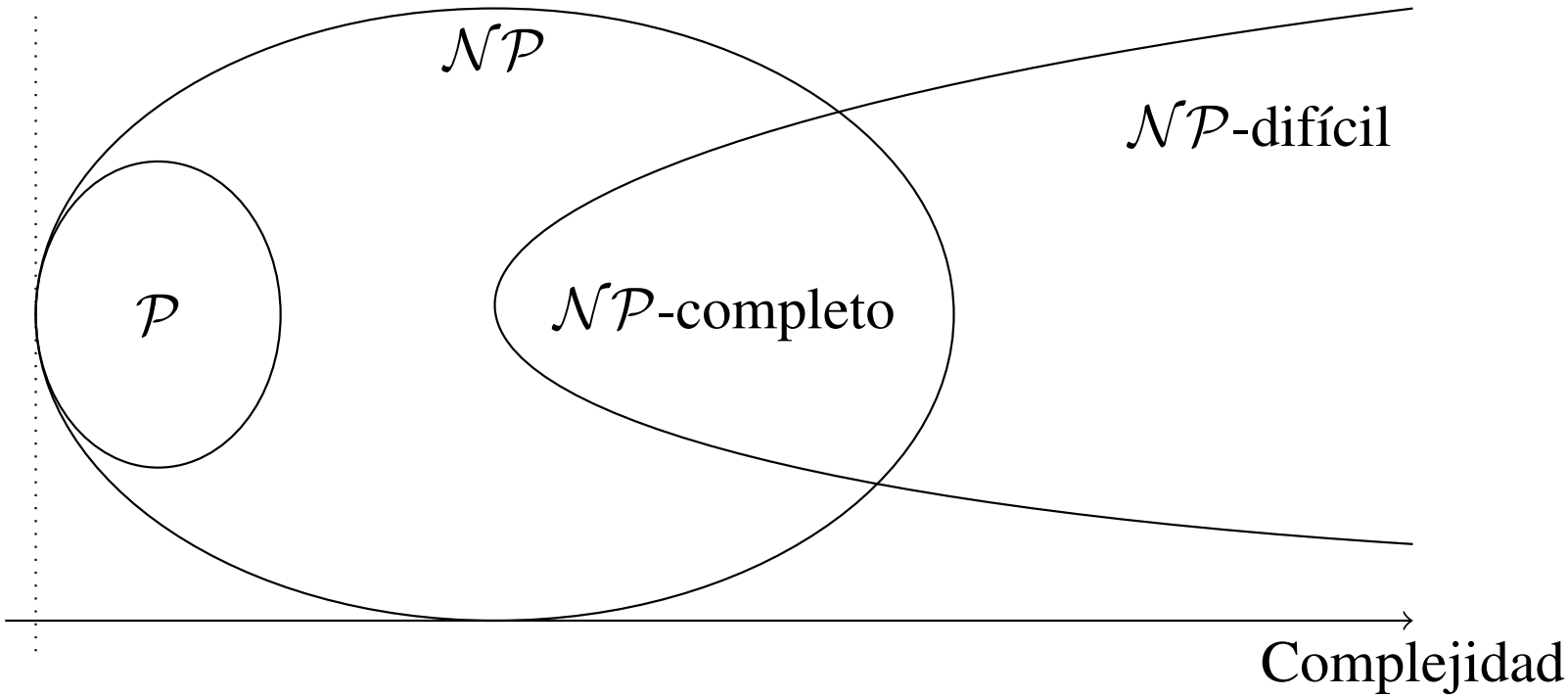
Proposición (11)

Sea el problema $X \in \mathcal{NP}$-completo. X es resoluble en tiempo polinomial si y solo si $\mathcal{P} = \mathcal{NP}$.

Por lo cual si existe algún problema en $\mathcal{NP}$ que no se pueda resolverse en tiempo polinomial, entonces no hay problema $\mathcal{NP}$-completo que se pueda resolver en tiempo polinomial.

Diagrama de clases

Las clases se pueden representar mediante un diagrama de Euler.



Problema de Satisfacibilidad de Circuito (C-SAT)

Problema inicial en $\mathcal{NP}$-completo

Para probar que un problema es $\mathcal{NP}$-completo, se debe mostrar que el problema *puede codificar cualquier problema* en $\mathcal{NP}$.
Se debe determinar un problema inicial en $\mathcal{NP}$-completo.

Un *circuito* es un dispositivo físico que implementa las operaciones lógicas: $\wedge$ (AND), $\vee$ (OR), y $\neg$ (NOT); denominadas como *puertas lógicas*.
Una expresión Booleana (ej. una instancia de 3-SAT) puede implementarse con un circuito mediante un árbol dirigido acíclico. Ejemplo:

Dado un circuito se debe determinar si existe una asignación de valores de las entradas tal que determina una salida de valor 1.

C-SAT es $\mathcal{NP}$-completo (1/2)

Proposición (13) (Cook y Levin, 1971)

C-SAT es $\mathcal{NP}$-completo

Prueba (bosquejo)

Requiere que para todo $X \in \mathcal{NP}$, $X \leq_P$ C-SAT. (*)

Cualquier algoritmo que tiene una entrada de n de bits y produce una respuesta Si/No puede implementarse mediante un circuito con una entrada de n y una salida de un bit.

El circuito es equivalente al algoritmo cuando su salida de valor 1 corresponde con las entradas y una salida Si en el algoritmo. Si el algoritmo emplea una cantidad polinomial en n de pasos, entonces el circuito que lo implementa tiene un tamaño polinomial equivalente de puertas lógicas.

Para demostrar (*). Dada una entrada s, se busca determinar si $s \in X$ usando una caja negra que resuelve instancias de C-SAT. Dado que $X \in \mathcal{NP}$, entonces X tiene un certificador eficiente B.

...

C-SAT es $\mathcal{NP}$-completo (2/2)

Prueba (cont.)

Para determinar que una entrada s de tamaño n cumple $s \in X$ se debe determinar si ¿Existe un certificado t de tamaño $p(n)$ tal que $B(s,t) = \text{Si}$? El certificador $B(s,t)$ es un algoritmo que tiene tamaño de entrada $n + p(n)$ bits, correspondientes a s y t, respectivamente.

Al algoritmo se lo puede implementar con un circuito de tamaño polinomial en $n + p(n)$ entradas, equivalentes.

Las primeras n entradas de B están fijas en la codificación del circuito, y las restantes $p(n)$ entradas de B se corresponden con las entradas del circuito.

Entonces $s \in X$ si y solo si existe una forma de configurar los bits de entrada al circuito tal que este produce una salida de 1; es decir si el circuito es satisfacible. Esto establece que $X \leq_P$ C-SAT.

□

Determinación de otros problemas en $\mathcal{NP}$-completo

Proposición (14)

Si Y es un problema $\mathcal{NP}$-completo y X es un problema en $\mathcal{NP}$ tales que $Y \leq_P X$, entonces X es $\mathcal{NP}$-completo.

Prueba.

Dado que $X \in \mathcal{NP}$ se cumple la propiedad (i) de la definición de $\mathcal{NP}$-completo y solo se necesita verificar la propiedad (ii).
Sea Z cualquier problema en $\mathcal{NP}$. Se tiene que $Z \leq_P Y$, por $\mathcal{NP}$-completitud de Y, e hipótesis $Y \leq_P X$. Por relación transitiva, se tiene que $Z \leq_P X$. □

Mientras que obtener un primer problema en $\mathcal{NP}$-completo lleva mucho esfuerzo (Prop. (13)), probar que problemas adicionales pertenecen a $\mathcal{NP}$-completo sólo requiere una reducción desde un problema que ya se sabe que es $\mathcal{NP}$-completo (Prop. (14)).

3-SAT es $\mathcal{NP}$-completo (1/3)

Proposición (15)

3-SAT es $\mathcal{NP}$-completo

Prueba.
3-SAT pertenece a $\mathcal{NP}$. Por lo que se probará que 3-SAT es $\mathcal{NP}$-completo a partir de que C-SAT $\leq_P$ 3-SAT.
Dada una instancia de C-SAT se construye una instancia de SAT en la que cada cláusula contiene a lo sumo tres términos. Luego se convierte esta instancia a una en que cada cláusula contiene exactamente tres términos (una instancia 3-SAT).
(...)

3-SAT es $\mathcal{NP}$-completo (2/3)

Dado un circuito con cada entrada v (nodo fuente) denominada por la variable x_v. Se considera la codificación de las puertas lógicas en expresiones SAT como

- $\neg$ Establece para el nodo de entrada u y el de salida v: $x_v := \overline{x_u}$. Para codificarlo en SAT se establecen las cláusulas $(x_v \vee x_u)$ y $(\overline{x_v} \vee \overline{x_u})$. (†)
- $\vee$ Establece para los nodos de entrada u y w, y el de salida v: $x_v := (x_u \vee x_w)$. Para codificarlo en SAT se establecen las cláusulas $(x_v \vee \overline{x_u})$, $(x_v \vee \overline{x_w})$, y $(\overline{x_v} \vee x_u \vee x_w)$.
- $\wedge$ Establece para los nodos de entrada u y w, y el de salida v: $x_v := (x_u \wedge x_w)$. Para codificarlo en SAT se establecen las cláusulas $(\overline{x_v} \vee x_u)$, $(\overline{x_v} \vee x_w)$, y $(x_v \vee \overline{x_u} \vee x_w)$.

Luego para un nodo fuente v con valor constante, se agrega una cláusula con la variable única x_v o $\overline{x_v}$, que obligue a tomar dicho valor. (‡)
Para un nodo de salida v, se agrega una variable x_v, la que se requiere que tome valor 1.

3-SAT es $\mathcal{NP}$-completo (3/3)

Finalmente se debe mostrar que la instancia de 3-SAT construida es equivalente a la instancia de C-SAT.
Por una parte, se supone que el circuito es satisfacible. Entonces la asignación de los nodos fuentes en el circuito y su propagación hasta el nodo de salida permite establecer un correlato satisfacible en la asignación y evaluación de las expresiones en la instancia de 3-SAT.
Por la otra parte, se supone que la instancia SAT es satisfacible.
Sea una asignación de valores satisfacible para la instancia SAT. Entonces se tiene que dichos valores constituyen una asignación satisfacible.
Las cláusulas SAT aseguran que los valores asignados a los nodos del circuito son iguales a que el circuito determina para los nodos. En particular, se asigna un valor 1 a la salida, por lo que la asignación a las entradas satisface el circuito.
Para finalizar, es necesario convertir esta instancia de SAT en una instancia equivalente en la que cada cláusula tiene exactamente tres literales.

□

Problemas en $\mathcal{NP}$-completo

A partir de la reducciones en tiempo polinomial de los problemas vistas anteriormente:

3-SAT $\leq_P$ Conjunto Indep. $\leq_P$ Cober, por Vértices $\leq_P$ Cober. en Conjunto
Conjunto Independiente $\leq_P$ Empaque en Conjunto

y la Prop. (14).

Se puede determinar que todos ellos pertenecen a $\mathcal{NP}$-completo.

Procedimiento para probar que un nuevo problema pertenece a $\mathcal{NP}$-completo

Dado un nuevo problema X el procedimiento para probar que pertenece a $\mathcal{NP}$-completo consiste en:

1. Probar que X pertenece a $\mathcal{NP}$.
2. Seleccionar un problema Y que se conoce que pertenece a $\mathcal{NP}$-completo.
3. Probar $Y \leq_P X$.

Si el paso (3.) consiste en transformar una instancia dada de Y en una única instancia de X con la misma respuesta, entonces este paso se puede describir como:
A partir de una instancia arbitraria s_Y del problema Y se requiere construir, en tiempo polinomial, una instancia s_X del problema X que satisface:

a. Si s_Y es una instancia Si de Y, entonces s_X es una instancia Si de X.
b. Si s_X es una instancia Si de X, entonces s_Y es una instancia Si de Y.

Categorías de problemas computacionales difíciles

Los problemas se pueden categorizar según su estructura:

1. Problemas de empaque: Conjunto Independiente y Empaque en Conjunto
2. Problemas de cobertura: Cobertura por Vértices y Cobertura en Conjunto
3. Problemas de particionado: Relación 3-Dimensional y Coloreo de Grafos
4. Problemas de secuenciado: Ciclo Hamiltoniano y Vendedor Viajero
5. Problemas numéricos: Suma en Subconjuntos y Mochila
6. Problemas de satisfacción de restricciones: C-SAT, SAT, Sudoku y n-Reinas