Matemáticas Discretas

Introducción a la complejidad computacional

Gabriel Diéguez
gsdieguez@ing.puc.cl

Fernando Suárez
fsuarez1@ing.puc.cl

Departamento de Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica de Chile

9 de Diciembre de 2019

Objetivos

1. Determinar la dificultad relativa de problemas computacionales, basando sus argumentos en técnicas de complejidad computacional.

Contenidos

Motivación

Una pregunta fundamental en computación es qué tan difícil es un problema particular.

- Ordenar un arreglo.
- Encontrar un camino en un grafo.
- Contestar una consulta en una base de datos.
- Optimizar la cantidad de materiales para alguna tarea.
- Asignar los cursos de la universidad.
- ...

En un capítulo anterior estudiamos herramientas para contestar esta pregunta para algoritmos particulares.

Motivación

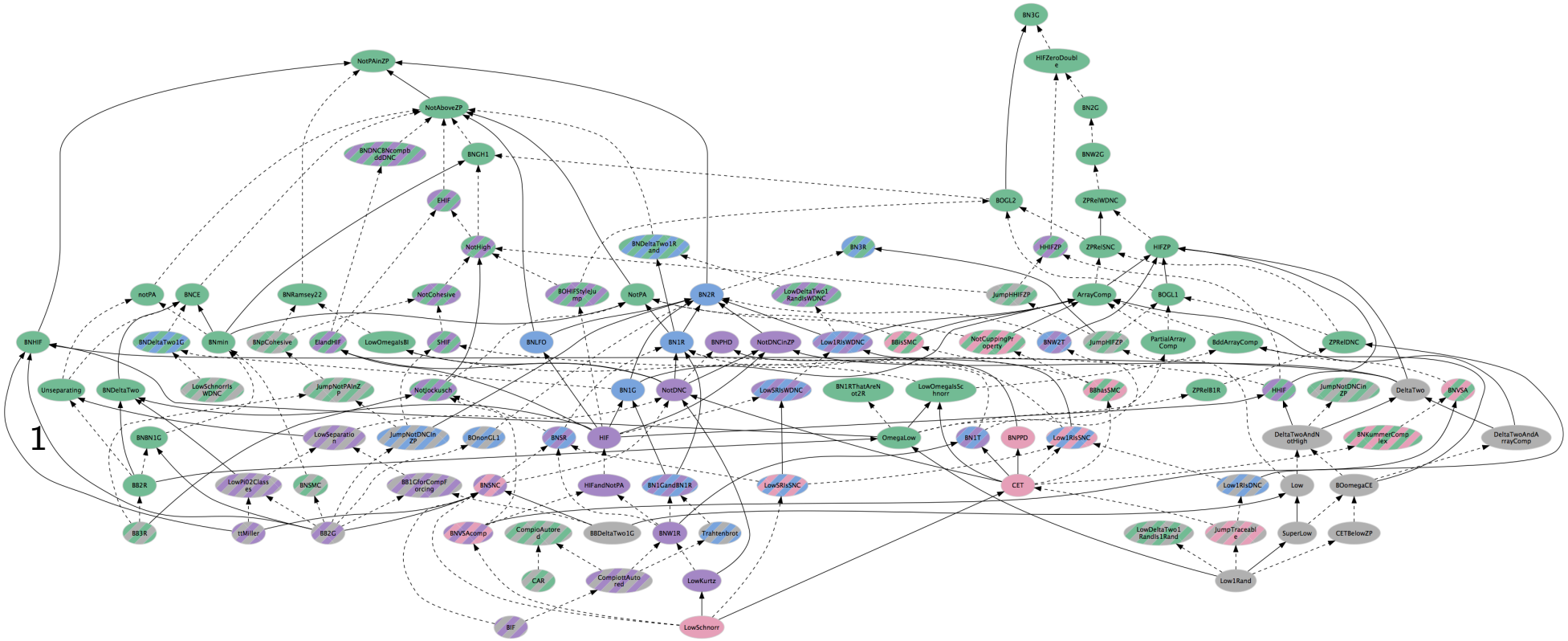
Más que en un algoritmo en particular, estaremos interesados en todos los posibles algoritmos que resuelven un problema particular. De esta forma podemos caracterizar un problema.

ą Tomaremos estos problemas y los encasillaremos en clases !

Motivación

£Cuántas clases de complejidad existen?

- PTIME
- NP
- co-NP
- EXPTIME
- PSPACE
- #PTIME
- NC
- BQP
- Problemas indecidibles
- etc...



[1]https://complexityzoo.uwaterloo.ca/Complexity_Zoo

Motivación

En este capítulo veremos una breve introducción a las siguientes clases:

- PTIME
- NP
- co-NP
- EXPTIME
- PSPACE
- #PTIME
- NC
- L
- BQP
- Problemas indecidibles
- etc...

Problemas

Existen 4 tipos de problemas fundamentales en computación:

- Decisión
- Búsqueda
- Evaluación
- Optimización

Todos son equivalentes entre sí, nos centraremos sólo en sus versiones de decisión.

Problemas de decisión

Definición (informal)

Son los problemas para los cuales sus respuestas posibles son SI o NO.

Definición (formal)

Un **problema de decisión** π se compone de un conjunto de instancias I_π y un lenguaje $L_\pi \subseteq I_\pi$, y se define como:

Dado un elemento $w \in I_\pi$, determinar si $w \in L_\pi$.

Problemas de decisión

Ejemplos

PRIMO: $I_{\text{PRIMO}} = \mathbb{N} - \{0, 1\}$, $L_{\text{PRIMO}} = \{p \in \mathbb{N} \mid p \text{ es primo}\}$.

EULERIANO: $I_{\text{EULERIANO}} = \{G \mid G \text{ es un grafo conexo}\}$,
$L_{\text{EULERIANO}} = \{G \mid G \text{ es un grafo y tiene un ciclo euleriano}\}$.

SAT: $I_{\text{SAT}} = L(P)$, $L_{\text{SAT}} = \{\varphi \in L(P) \mid \varphi \text{ es satisfacible}\}$.

Problemas de decisión

Diremos que un algoritmo A resuelve un problema de decisión π si para cada input $w \in I_\pi$, el algoritmo A responde SI cuando $w \in L_\pi$, y responde NO cuando $w \in I_\pi - L_\pi$ (en este caso diremos que $w \notin L_\pi$).

Ejercicio

Encuentre algoritmos para los problemas anteriores. £Qué complejidad tienen?

Clases de complejidad

Definición

La clase de complejidad DTIME($T(n)$) es el conjunto de problemas para los cuales existe un algoritmo que lo resuelve de complejidad $O(T(n))$:

$$\text{DTIME}(T(n)) = \{\pi \mid \pi \text{ es un problema de decisión para el cual existe un algoritmo } A \text{ que lo resuelve de complejidad } O(T(n))\}$$

£A qué clases de complejidad pertenecen los problemas anteriores?

La clase PTIME

Definición

La clase de complejidad PTIME o simplemente P se define como

$$P = \bigcup_{k=0}^{\infty} \mathsf{DTIME}(n^k)$$

La clase PTIME contiene a todos los problemas que pueden resolverse en tiempo polinomial. En general diremos que estos problemas son *tratables*.

Ejemplo

$\mathrm{EULERIANO} \in P$.

La clase NP

En el caso de SAT, encontramos un algoritmo exponencial que lo resuelve. . . pero si tenemos la valuación, es fácil chequear si satisface a la fórmula. Esto motiva la siguiente definición:

Definición (informal)

La clase de complejidad NP contiene a todos los problemas de decisión para los cuales es posible verificar una solución al problema de forma eficiente. Sin embargo, no necesariamente encontrar la solución sea fácil.

Definición (formal)

La clase de complejidad NP contiene a todos los problemas de decisión π para los cuales se cumple lo siguiente:

Si $w \in L_\pi$, entonces existe un *certificado* $c(w)$ de tamaño polinomial, tal que existe un algoritmo que usando $c(w)$ puede determinar en tiempo polinomial si $w \in L_\pi$.

La clase NP

£Por que se llama NP?

"Se cuenta el milagro, pero no el santo"

Luis Dissett

Saber si 95647 es compuesto es bastante complicado, pero nadie cuestiona lo elemental que es multiplicar 101 por 947.

La clase NP

£Qué problemas están en NP?

- CLIQUE
- SAT-CNF
- 3-COLOREABLE
- HAMILTIONIANO
- entre otros...muchos

Ejercicio

Demuestre que estos problemas están en NP.

La clase NP

El siguiente teorema nos permite establecer una primera cota para las clases P y NP

Teorema

$\mathrm{P} \subseteq \mathrm{NP}$.

Ejercicio

Demuestre el teorema.

Reducciones polinomiales

Hasta ahora sabemos un método para establecer que tan difíciles pueden llegar a ser los problemas (*upper bound*). Sin embargo, también existen métodos para establecer posibles cotas inferiores (*lower bound*).

Definición

Dados dos problemas de decisión $\pi = (I_\pi, L_\pi)$ y $\pi' = (I_{\pi'}, L_{\pi'})$. Diremos que π' se reduce desde π si dada una instancia $w \in I_\pi$ existe una función $A : I_\pi \to I_{\pi'}$ polinomial en $|w|$ tal que:

$$w \in L_\pi \leftrightarrow A(w) \in L_{\pi'}$$

En este caso diremos que π' es **por lo menos tan difícil** como π y lo denotaremos como $\pi \propto \pi'$.

Reducciones polinomiales

Recordando:

Definición

Decimos que una fórmula φ está en **forma normal conjuntiva (CNF)** si es una conjunción de disyunciones de literales; o sea, si es de la forma

$$C_1 \wedge C_2 \wedge \ldots \wedge C_k$$

donde cada C_i es una disyunción de literales, $C_i = (l_{i1} \vee \ldots \vee l_{ik_i})$

- Una disyunción de literales se llama una **cláusula**.
  - Los C_i anteriores son cláusulas.
- Una fórmula en CNF es una conjunción de cláusulas.

Ejemplo

$(p \vee \neg q) \wedge (\neg p \vee \neg r \vee s) \wedge (\neg r \vee s)$

Reducciones polinomiales

Recordemos también que una conjunción de fórmulas es equivalente al conjunto de ellas. Luego, podemos considerar que una fórmula en CNF es un conjunto de cláusulas:

$$C_\varphi = \{C_1, C_2, \ldots, C_k\}$$

Más aún, cada cláusula C_i puede verse como un conjunto de literales.

Ejemplo

$\varphi = (p \vee \neg q) \wedge (\neg p \vee \neg r \vee s) \wedge (\neg r \vee s)$
$C_\varphi = \{\{p, \neg q\}, \{\neg p, \neg r, s\}, \{\neg r, s\}\}$

Reducciones polinomiales

Definición

Decimos que una fórmula φ está en **3-CNF** si es una conjunción de cláusulas de exactamente 3 literales cada una.

- En general aplicamos la restricción de 3 literales al conjunto que representa a cada cláusula.
- Por lo tanto, no pueden repetirse literales en la misma cláusula.

Ejemplo

$\varphi = (p \vee q \vee \neg q) \wedge (\neg p \vee \neg r \vee s) \wedge (\neg r \vee s \vee \neg s)$
$C_\varphi = \{\{p, q, \neg q\}, \{\neg p, \neg r, s\}, \{\neg r, s, \neg s\}\}$

Reducciones polinomiales

Definimos $\mathrm{SAT\text{-}3CNF}$ como:

Definición

$$I_{\mathrm{SAT\text{-}3CNF}} = \varphi \in L(P) \text{ en 3-CNF}$$
$$L_{\mathrm{SAT\text{-}3CNF}} = \{\varphi \in L(P) \text{ en 3-CNF} \mid \varphi \text{ es satisfacible}\}.$$

Ejercicio

Demuestre que $\mathrm{SAT\text{-}CNF} \propto \mathrm{SAT\text{-}3CNF}$.

En este caso diremos que $\mathrm{SAT\text{-}3CNF}$ es **por lo menos tan difícil** como $\mathrm{SAT\text{-}CNF}$.

Reducciones polinomiales

Ejercicio

Demuestre que $\text{SAT-CNF} \propto \text{SAT-3CNF}$.

La reducción la haremos desde SAT-CNF. Dado un conjunto $C = \{c_1, \ldots, c_m\}$ de cláusulas, construiremos un nuevo conjunto C' de cláusulas de tamaño 3 tal que C sea satisfacible si y sólo si C' sea satisfacible. Sea $c_i \in C$ definida por los literales $\{l_1, \ldots, l_k\}$, el procedimiento dependerá del tamaño de la cláusula:

- $k = 1$.
  En este caso $c_i = \{l_1\}$. Agregamos variables adicionales $\{x_1, x_2\}$ y construimos las siguientes cláusulas
  $C'_i = \{\{l_1, x_1, x_2\}, \{l_1, \bar{x_1}, x_2\}, \{l_1, x_1, \bar{x_2}\}, \{l_1, \bar{x_1}, \bar{x_2}\}\}$.
- $k = 2$.
  En este caso $c_i = \{l_1, l_2\}$. Agregamos una variable adicional x_1 y construimos las siguientes cláusulas $C'_i = \{\{l_1, l_2, x_1\}, \{l_1, l_2, \bar{x_1}\}\}$.

Ejercicio

Demuestre que $\text{SAT-CNF} \propto \text{SAT-3CNF}$.

- $k = 3$.
  En este caso $c_i = \{l_1, l_2, l_3\}$ ya está en 3CNF y no la alteramos.
  Luego $C'_i = \{c_i\}$.
- $k > 3$.
  En este caso $c_i = \{l_1, \ldots, l_k\}$. Agregamos $k - 3$ variables adicionales $\{x_1, \ldots, x_{k-3}\}$ y construimos cláusulas

$$C'_i = \{\{l_1, l_2, x_1\}, \{\bar{x_1}, l_3, x_2\}, \{\bar{x_2}, l_4, x_3\}, \ldots, \{\bar{x_{k-3}}, l_{k-1}, l_k\}\}$$

Finalmente construimos $C' = \bigcup C'_i$. Además, notemos que la construcción propuesta toma tiempo polinomial acotado por $n \cdot m$. Por otro lado, es posible demostrar que la reducción es correcta.

Reducciones polinomiales

Ejercicio

Demuestre que $\text{SAT-CNF} \propto \text{SAT-3CNF}$.

($\Rightarrow$) Sea C satisfacible, construiremos una valuación que satisface todas las cláusulas de C', para esto usamos la misma valuación que satisface C pero la extendemos para las variables auxiliares. Cuando $k \in \{1, 2, 3\}$ podemos asignar valores de verdad arbitrarios. En el caso en que $k > 3$, sabemos que existe un $l_j \in \{l_1, \ldots, l_k\}$ que debe ser igual a 1. Si $j = 1$ o $j = 2$, entonces la valuación debe hacer 0 a todas las variables auxiliares. Si $j = k - 1$ o $j = k$, entonces la valuación debe hacer 1 a todas las variables auxiliares. Si $k \in \{3, \ldots, k - 2\}$, entonces la valuación debe hacer 1 a todo x_p con $p < j - 1$ y 0 al resto.

($\Leftarrow$) Por contrapositivo, suponemos que C no es satisfacible, por lo tanto existe por lo menos un c_i que no es satisfacible y todos sus l_i son 0. Luego, por construcción todo C'_i es falso en el caso en que los l_i son 0, por lo tanto C' no es satisfacible.

Reducciones polinomiales

Teorema

$\propto$ es un preorden (refleja y transitiva).

Ejercicio

Demuestre el teorema.

NP-hard

El hecho de que $\propto$ sea un preorden nos permite establecer una jerarquía en cuanto al a dificultad de los problemas. Esto motiva la siguiente definición:

Definición

Sea π un problema de decisión. Diremos que π es NP-hard si para todo $\pi' \in \mathrm{NP}$ se tiene que $\pi' \propto \pi$

En otras palabras, los problemas NP-hard son por lo menos tan difíciles como todos los problemas en NP !

NP-hard

El único problema con la definición anterior es que no nos dice nada sobre las cotas superiores de los problemas. La siguiente definición logra capturar la verdadera complejidad de la clase NP.

Definición

Sea π un problema de decisión. Diremos que π es NP-completo si:

- $\pi \in \mathrm{NP}$
- π es NP-hard

Dado que cualquier problema en NP se puede reducir a uno NP-completo, podemos concluir que los problemas NP-completos son los más dificiles de la clase NP.

NP-completitud

Para demostrar que un problema es NP-completo basta con reducir desde otro problema NP-completo!
El problema es que hasta ahora no hemos mencionado ningún problema NP-completo...

Teorema de Cook

SAT-CNF es NP-completo

La demostración se escapa de los contenidos de este curso :c.
Sin embargo, podemos utilizar este teorema para demostrar que otros problemas son NP-completos c:

NP-completitud

Definición

$$I_{\text{CLIQUE}} = \text{Todos los grafos no dirigidos.}$$
$$L_{\text{CLIQUE}} = \{(G, k) \mid G \text{ tiene un clique de tamaño } k \wedge k > 2\}.$$

Ejercicio

Demuestre que CLIQUE es NP-completo.

Ejercicio

Demuestre que CLIQUE es NP-completo.

- CLIQUE $\in$ NP: Demostrado al principio del capítulo.
- CLIQUE es NP-hard: La reducción la haremos desde SAT-3CNF. Sea C un conjunto de cláusulas en 3CNF, construiremos un grafo G tal que C sea satisfacible si y sólo si G tiene un clique. Sea m el número de cláusulas, consideremos el siguiente procedimiento:
  1. Definimos m conjuntos independientes (uno por cada cláusula),
  2. Por cada cláusula y literal l_i, agregamos a v_{l_i} a su conjunto respectivo.
  3. Agregamos aristas entre todos los vértices de distintos conjuntos, a excepción de los pares de la forma $\{l, \overline{l}\}$.
  4. No conectamos vértices del mismo conjunto independiente.

Ejercicio

Demuestre que CLIQUE es NP-completo.

Es claro que la reducción puede llevarse a cabo en tiempo polinomial sobre la cantidad de proposiciones. Además, notemos que la reducción es correcta.

($\Rightarrow$) Sea C satisfacible por una valuación σ, podemos construir el siguiente clique $K = \{v_l \mid \sigma(l) = 1\}$. Es claro que todos los vértices de K deben estar conectados, ya que si no violarian la regla 3 de la reducción[2].

($\Leftarrow$) Sea G el grafo construído desde C. Si G tiene tiene un clique K de tamaño m, definimos la valuación σ tal que $\sigma(l) = 1$ si y sólo si $v_l \in K$. Dado, que K tiene exactamente m vértices de conjuntos distintos, la valuación debe satisfacer a cada cláusula por separado.

[2]Notar que una valuación no puede satisfacer a l Y $\bar{l}$.

NP-completitud

£Qué sabemos hasta ahora?

- $\mathrm{P} \subseteq \mathrm{NP}$
- NP-completo $\subseteq \mathrm{NP}$
- $\mathrm{SAT\text{-}CNF} \in \mathrm{NP}$-completo
- $\textsc{Clique} \in \mathrm{NP}$-completo

Si pudiésemos resolver $\textsc{Clique}$ o $\mathrm{SAT\text{-}CNF}$ en tiempo polinomial £Qué podríamos concluir?

El problema abierto más importante en ciencias de la computación

$$\mathrm{P} \stackrel{?}{=} \mathrm{NP}$$

"If the solution to a problem can be quickly verified by a computer, can the computer also solve that problem quickly?"

Wikipedia.

El problema abierto más importante en ciencias de la computación

$$\mathrm{P} \stackrel{?}{=} \mathrm{NP}$$

"Aside from being an important problem in computational theory, a proof either way would have profound implications for mathematics, cryptography, algorithm research, artificial intelligence, game theory, multimedia processing, philosophy, economics and many other fields"

Wikipedia.

Matemáticas Discretas

Introducción a la complejidad computacional

Gabriel Diéguez
gsdieguez@ing.puc.cl

Fernando Suárez
fsuarez1@ing.puc.cl

Departamento de Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica de Chile

9 de Diciembre de 2019