

Complejidad de un Problema.

1 Presentación y definiciones

1.1 Problema del Vendedor Viajero

Para ilustrar este capítulo se va a trabajar en base al problema de teoría de grafos ampliamente conocido: El Problema del Vendedor Viajero¹.

Dado un grafo completo con “n” nodos, donde cada arco tiene asociado un largo, se busca un ciclo hamiltoniano de largo mínimo del grafo. Un algoritmo trivial sería: Enumerar todos los ciclos hamiltonianos del grafo y calcular su largo, de forma tal de poder determinar el de menor largo. Lamentablemente este algoritmo es imposible de llevar a cabo en la práctica, aún considerando un número reducido de nodos: Un grafo completo con n nodos posee $\frac{(n-1)!}{2}$ ciclos hamiltonianos. Por ejemplo para $n = 20$ se requeriría alrededor de 19 siglos de cálculo en un computador capaz de generar 1.000.000 de ciclos hamiltonianos por segundo. Se puede acelerar la resolución usando un algoritmo tipo branch and bound, sin embargo todos los algoritmos exactos conocidos actualmente requieren un tiempo de cálculo demasiado importante como para que sea razonable el aplicarlos, de un punto de vista práctico, cuando el tamaño del grafo se vuelve un poco elevado. Vamos a ver que este problema no es el único para el cual no se conoce un “buen” algoritmo de resolución exacta y que parece casi imposible el encontrar uno algún día (no significa que a priori este no exista). Veamos algunas definiciones previas.

2 Tamaño de una Instancia

Para definir el tamaño de una instancia se requiere tomar en cuenta todos los factores. Por ejemplo, en el caso de una instancia para el problema del camino más corto se debe tomar en cuenta no solo el número de nodos del grafo sino también de las distancias entre los pares de nodos. La descripción de una instancia de un problema puede ser visto como una cadena de caracteres que pertenecen a un alfabeto fijo. El largo de esta cadena de caracteres definirá el tamaño de la instancia. El alfabeto binario permite una codificación razonable. De esta forma representar un número entero k positivo con esta codificación requerirá $\text{round}(\log_2(k + 1))$ bits; representar un grafo de n nodos por su matriz

¹En inglés Travelling Salesman Problem

de adyacencia necesitará tantos bits como pares de nodos, es decir n^2 . En este sistema binario, el tamaño de la instancia es el número de bits necesarios para representar todos los datos que permiten definir la instancia, tratándose ya sea de números, conjuntos, grafos u otra estructura. Se adopta esta codificación para los estudios de complejidad.

3 Máquina de Turing y complejidad de un algoritmo

Para definir la noción de complejidad de un algoritmo se define primero lo que es una *Máquina de Turing determinista de una banda*, esta máquina está compuesta de tres partes:

- Un conjunto de estados de la máquina, constituyendo una “unidad central” que controla el conjunto de ruteo del programa, con la ayuda de una “tabla” que contiene la descripción de las operaciones a realizar.
- de una banda infinita constituida de casillas numeradas $\dots, -3, -2, -1, 0, 1, 2, \dots$ sobre la cual están inscritos los datos y donde serán inscritos los cálculos intermedios y los resultados finales.
- De una cabeza de lectura-escritura susceptible de leer y de modificar las informaciones que figuran sobre la banda, de acuerdo a las instrucciones entregadas por la “unidad central”.

El comportamiento de la máquina de Turing depende de la tabla almacenada en la unidad central y de los símbolos leídos en las casillas de la banda. En cada paso, la cabeza de lectura-escritura lee el símbolo contenido en la casilla de la banda sobre la cuál está apuntando. En función de este símbolo y del estado actual de la máquina, la tabla indica a la cabeza lo que debe escribir en la casilla sobre la que está apuntando, determina el desplazamiento de la cabeza (una casilla en un sentido o en el otro, a menos que reste inmóvil) y define el estado siguiente de la máquina.

El siguiente ejemplo ilustra el funcionamiento de una máquina de Turing y la modelización de un programa por este tipo de máquina, esto nos permite dar una definición precisa de lo que son los algoritmos polinomiales. La máquina está constituida:

1. De un conjunto finito de estados Q compuesto de un estado inicial q_0 , de dos estados finales q_s y q_e (s de éxito y e de fracaso; si se aplica esta máquina de Turing a un problema de reconocimiento, es decir a un problema donde uno hace una pregunta y cuya respuesta es “si” o “no”, el éxito se podrá interpretar como una respuesta “si” y el fracaso como un “no”). Se incorporan además los estados q_1 y q_2 .

2. De un conjunto finito de valores que se pueden leer o escribir en las casillas de la banda, que contienen una palabra de $\{0, 1\}^*$ (si A representa un alfabeto, es decir un conjunto de símbolos, A^* indica el conjunto de palabras que se pueden escribir a partir de A , es decir toda cadena de símbolos que pertenece a A) y un símbolo especial que se llamará *blanco* b .
3. De una función de transición que en un estado distinto de q_s y q_e tiene asociada una tripleta cuyo primer elemento indica el nuevo estado de la máquina, el segundo caracter reemplazará al caracter leído, el tercero indica el sentido de desplazamiento de la cabeza de la lectura (desplazamiento de a lo mas una casilla: se denotará por $+1$ un desplazamiento hacia la derecha, -1 un desplazamiento hacia la izquierda, y 0 la ausencia de desplazamiento); la máquina se detiene en alguno de los estados q_s o q_e .

La tabla indica las transiciones de la máquina (o programa) considerada. El índice de la línea corresponde al estado de la máquina, el índice de la columna al caracter leído sobre la banda.

| | 0 | 1 | b |
|-------|----------------|----------------|----------------|
| q_0 | $(q_0, 0, +1)$ | $(q_0, 1, +1)$ | $(q_1, b, -1)$ |
| q_1 | $(q_2, b, -1)$ | $(q_e, b, 0)$ | |
| q_2 | $(q_s, b, -1)$ | $(q_e, b, 0)$ | $(q_e, b, 0)$ |

El efecto de ese programa es el siguiente: el dato es una cadena X de símbolos pertenecientes a $\{0, 1\}$. La cadena X está localizada sobre la banda, un símbolo por casilla, el primero en la posición 1, el último en la posición $|X|$. Las otras casillas de la banda contienen inicialmente b . La máquina está inicialmente en el estado q_0 , y la cabeza de lectura-escritura está ubicada sobre la casilla número 1. El cálculo se realiza etapa por etapa. Es fácil ver que el estado q_0 consiste a desplazarse hacia la derecha en la búsqueda de un blanco, o sea al final de la palabra, sin cambiar nada. Cuando se encontró el primer blanco, se cambia de estado y se pasa a q_1 devolviéndose a una casilla anterior: Se apunta, a partir de ahora, sobre el último bit de la palabra. Si este es un "1", se detiene en el estado q_e : fracaso; si se lee un "0" se pasa al estado q_2 y se devuelve una casilla más (debe notarse que no es posible leer un blanco). Si el nuevo bit es un "0" se detiene en el estado q_s : la respuesta será "si"; en los otros dos casos se alcanza el estado final q_e : la respuesta es "no". Finalmente, se ve que esta máquina modela un programa que prueba que si una palabra de $\{0, 1\}^*$ se termina por un 00: esta es una condición necesaria y suficiente para que sea reconocido por la máquina, o para que la máquina termine el cálculo en el estado q_s .

Un problema de reconocimiento se puede traducir en términos de la máquina de Turing determinista (una banda). Por ejemplo, la máquina de Turing anterior está asociada al siguiente problema: dado un entero $N > 0$, existe un entero m tal que $N = 4m$?. La codificación asociada a una instancia del problema es simple, ya que basta con dar una representación binaria a N . La máquina de Turing definida se termina para todo dato (siguiendo los elementos iniciales en

la banda): el cálculo se termina en el estado “éxito” si los dos bits a la derecha son 0, ella se termina en “fracaso” sino.

Ahora se puede dar una definición precisa de la complejidad de la máquina M , o del algoritmo modelado por M . Si, para todo dato, se alcanza un estado final (q_e ó q_s) en un número finito de etapas, se puede definir un “tiempo de cálculo” o complejidad como el número máximo de pasos efectuados por M para poder tratar cualquier dato, de tamaño fijo del problema. Este número de pasos o este tiempo de cálculo $T_M(n)$ es una función del tamaño $n = |X|$ del dato X . Si existe un polinomio p tal que $T_M(n) \leq p(n)$, se habla de una máquina de Turing polinomial y se dice que el algoritmo modelado por M es polinomial (lo que no significa que su complejidad se pueda expresar por un polinomio con respecto al tamaño de los datos, sino solamente que se puede acotar por tal polinomio; así un algoritmo cuya complejidad vale $\log_2 n$, donde n es el tamaño de los datos, es polinomial). Un algoritmo cuya complejidad no es acotable por un polinomio con respecto al tamaño de los datos se dice *exponencial*, aunque su complejidad no se exprese a través de una exponencial en el sentido matemático del término (por ejemplo, un algoritmo de complejidad $n!$, donde n es el tamaño del dato, es exponencial: la función factorial no es acotable por un polinomio en n , sin ser ella misma una función exponencial). A veces se da el nombre de “buenos algoritmos” o de “algoritmos eficaces” a los algoritmos polinomiales; sin embargo, en la práctica sucede que algoritmos exponenciales muestran una mejor performance.

En la práctica, esta definición de complejidad de un algoritmo es reemplazada por una noción más fácil de manipular. Se sustituye el número de pasos efectuados por una máquina de Turing por el número de operaciones consideradas elementales efectuadas por un computador secuencial; se entiende por elemental una operación modelada en un número polinomial (con respecto al tamaño de los datos) de pasos de una máquina de Turing: Este es el caso por ejemplo de las cuatro operaciones aritméticas, de las diferentes operaciones de comparación y de asignación.

4 Problemas de Reconocimiento

Los problemas de reconocimiento se conocen también como problemas de decisión, es decir de enunciados para los cuales la respuesta es “si” o “no” y no a problemas de optimización. Existe una unión importante entre un problema de reconocimiento y problema de optimización, esto se ilustrará con el problema del vendedor viajero. El problema del vendedor viajero (TSP) se puede escribir de la siguiente manera (para simplificar el desarrollo se supondrá que los valores del grafo son enteros y positivos):

Nombre: TSP

Datos: Dado un grafo G de orden n , completo y valorizado con valores positivos

Objetivo: Determinar el largo mínimo de un ciclo hamiltoniano en el grafo G

El problema de reconocimiento (RTSP) asociado al TSP se puede enunciar como:

Nombre: RTSP

Datos: Dado un entero k , dado un grafo G de orden n , completo y valorizado con valores positivos

Pregunta: Existe en el grafo un ciclo hamiltoniano de largo inferior o igual a k ?

Es obvio que si se resuelve TSP para el grafo G , se resuelve inmediatamente RTSP. En consecuencia, el tamaño de las instancias de RTSP y de TSP difieren entre ellas sólo por el término $\text{round}(\log_2(k + 1))$, si existe un algoritmo polinomial (con respecto al tamaño de las instancias de TSP) para resolver TSP, entonces existe un algoritmo polinomial (con respecto al tamaño de las instancias de RTSP) para resolver RTSP. Suponga que se resuelve sucesivamente un número finito de instancias de RTSP, dándole valores enteros a k , disminuyendo de 1 a cada etapa. Partiendo de un valor inicial de k , para el cual se está seguro que la respuesta es “si”, el último valor de k para el cual la respuesta es “si” entrega el valor mínimo del ciclo hamiltoniano. Se puede realizar un procedimiento mejor como utilizar la dicotomía para encontrar el mínimo: Se comienza al igual que antes, dando un valor a k para el cual la respuesta es “si”, por ejemplo $n \cdot \text{val}_{\max}$ donde val_{\max} denota el valor más grande del grafo. Se resuelve RTSP asignando $k = \text{int}(\frac{\text{val}_{\max}}{2})$: si la respuesta es “si” se sigue con $k = \text{int}(\frac{\text{val}_{\max}}{4})$, sino con $k = \text{int}(\frac{3 \cdot \text{val}_{\max}}{2})$ y así sucesivamente. Así en las partes enteras cercanas el intervalo de búsqueda, disminuyen a la mitad y bastan alrededor de $\log_2(n \cdot \text{val}_{\max})$ resoluciones de instancias de RTSP para resolver la instancia asociada al TSP. Luego, todo algoritmo A_R de complejidad $c(A_R)$ que permite resolver RTSP define un algoritmo A_0 de complejidad del orden de $\log_2(n \cdot \text{val}_{\max}) c(A_R)$ que permite resolver TSP. O si val denota la función de evaluación de G , el tamaño de una instancia de TSP es del orden de $T_0 = \sum_{\{i,j\} \in G} \log_2(\text{val}(i,j) + 1)$, ya que se deben codificar los enteros $\text{val}(i,j)$ para todos los arcos (i,j) del grafo (se debe notar que esta cantidad es superior o igual a $\frac{n(n-1)}{2}$ superior a $\log_2 n$) y el tamaño de la instancia de RTSP es del orden de $T_R = \log_2(k + 1) + \sum_{\{i,j\} \in G} \log_2(\text{val}(i,j) + 1)$. Dándose cuenta que se está interesado sólo en las instancias de RTSP para las cuales $k < n \cdot \text{val}_{\max}$ (las otras instancias tienen la respuesta trivial “si”), es fácil mostrar el siguiente resultado: si existe un polinomio p en T_R que acota la complejidad de A_R , luego existe un polinomio q en T_0 que acota la complejidad de A_0 . Dicho de otra forma, si existe un algoritmo polinomial (con respecto al tamaño de las instancias de RTSP) para resolver RTSP, luego existe un algoritmo polinomial (con respecto al tamaño de las instancias de TSP) para resolver TSP.

Estos resultados muestran la unión entre RTSP y TSP: uno permite resolución en tiempo polinomial si y sólo si el otro también lo permite. Esta propiedad, general, nos permite interesarnos en los problemas de reconocimiento para obtener información de la dificultad de los problemas de optimización a los cuales éstos están asociados.

5 Clases P y NP; problemas NP-completos

Las nociones presentadas en esta sección se refieren a problemas de reconocimiento, sin embargo, como se mencionó anteriormente, los resultados obtenidos nos darán informaciones relacionadas con los problemas de optimización.

5.1 La clase P

Definición 5.1 *Un problema se dice polinomial si existe un algoritmo de complejidad polinomial que permite responder la pregunta del problema cualquiera sea el dato de éste. La clase P es el conjunto de todos los problemas de reconocimiento polinomiales.*

Se conocen varios problemas que son polinomiales. Por ejemplo los problemas de programación lineal, aunque no sea el algoritmo simplex que permite responder la pregunta. Sin embargo, no conocer un algoritmo polinomial que resuelva un problema dado no significa que no exista. Para tener en cuenta esta dificultad, se define una clase de problemas mas amplia, la clase NP.

5.2 La clase NP

Definición 5.2 *Un problema de reconocimiento está en la clase NP si, para toda instancia de ese problema, se puede verificar, en un tiempo polinomial con respecto al tamaño de la instancia, que una solución propuesta o adivinada permite afirmar que la respuesta es “si” para esta instancia.*

Se debe notar que no nos interesa la justificación de una respuesta negativa: se busca sólo verificar una respuesta “si”. Veamos el problema de vendedor viajero

Proposición 5.1 *RTSP está en la clase NP*

Prueba: Basta demostrar cómo verificar, en un tiempo polinomial, que una posible solución permite establecer la respuesta “si” para la instancia considerada. Pensemos que alguien nos dice que la respuesta de esta instancia es “si” y, para convencernos de este resultado presenta una manera de visitar las ciudades diciendo que es un ciclo hamiltoniano de largo inferior o igual a una cota k . Habría entonces que verificar dos cosas:

1. La estructura propuesta es un ciclo hamiltoniano
2. El largo de ese ciclo es inferior o igual a k

Para ello, verificamos primero que la secuencia S de nodos propuestos como solución contiene el número n de nodos: luego, se crea una tabla T con n casillas enumeradas por nodo e inicializada en 0; luego se recorren nuevamente los nodos de S y se incrementa de acuerdo a los valores contenidos en T : Al final, los valores de T entregan el número de ocurrencias de los nodos en S . Es fácil

deducir así si S es un ciclo hamiltoniano. Se puede también durante el recorrido calcular el largo de S ; comparándolo con k , se verifica de esta manera la parte 2. Todas estas verificaciones necesitan un número de operaciones $O(n)$ lo que es acotable por un polinomio $\log_2(k+1) + \sum_{\{i,j\} \in G} \log_2(\text{val}(i,j) + 1)$, el orden del tamaño de la instancia RTSP. Se pudo así verificar en un tiempo polinomial que una posible solución entrega correctamente la respuesta “si”. En consecuencia RTSP está en NP. Por otro lado, veamos un problema polinomial. Siempre es posible verificar que la respuesta es “si” en un tiempo polinomial, para cualquier instancia: Basta con resolver en un tiempo polinomial, el problema para la instancia considerada y ver si la respuesta es “si” o “no”. De ahí la siguiente proposición:

Proposición 5.2 *Proposición:* $P \subseteq NP$

Se han definido las clases P y NP . No se sabe si estas dos clases coinciden o si la inclusión de la clase P en la clase NP es estricta. De todas maneras estamos interesados en los problemas NP considerados como los más difíciles de esta clase, los problemas NP-completos

5.3 Problemas NP-Completos

Transformación polinomial

Una noción fundamental en la teoría de la NP-completitud es la de la *transformación polinomial*: dados dos problemas de decisión D_1 y D_2 , se cumplirá que $D_1 \prec D_2$ si se cumplen las siguientes condiciones:

1. Existe una aplicación f que transforme cualquier instancia I de D_1 en una instancia $f(I)$ de D_2 , y un algoritmo polinomial, con respecto al tamaño de I para calcular $f(I)$.
2. Hay una equivalencia entre los dos enunciados “ D_1 acepta la respuesta “si” para la instancia I ” y “ D_2 acepta la respuesta “si” para la instancia $f(I)$ ”.

Si $D_1 \prec D_2$ y si existe un algoritmo polinomial para resolver D_2 , luego existe un algoritmo polinomial para resolver D_1 . Para obtener la respuesta de la instancia I de D_1 , basta con transformar I polinomialmente en la instancia $f(I)$ de D_2 y resolver ésta, siempre polinomialmente: se resuelve I resolviendo $f(I)$. Se puede interpretar intuitivamente la relación \prec como que D_1 no es más difícil que D_2 . Esta es una noción transitiva, es decir si $D_1 \prec D_2$ y $D_2 \prec D_3$, luego $D_1 \prec D_3$.

Definición 5.3 *Un problema Q se dice NP-completo si pertenece a la clase NP y si, para todo problema Q' de la clase NP, se tiene que $Q' \prec Q$.*

Esta definición tiene varios aspectos importantes: si un problema NP-completo se comprueba polinomial, entonces $P = NP$; si Q y Q' son dos problemas de

la clase NP, si Q es NP-completo y si $Q \prec Q'$ entonces Q' es NP-completo. Finalmente, nótese que en el caso $P \neq NP$, los problemas NP-completos y los problemas polinomiales no involucran todos los problemas en NP; si $P \neq NP$ se puede mostrar por el contrario que existe un número infinito de clases de problemas de NP de dificultad intermedia con respecto a esas clases, P siendo la clase menos difícil y los problemas NP-completos la clase más difícil.

6 Consecuencias de la NP-completitud de un problema

Para qué sirve la demostración, larga y complicada de que un problema es NP-completo?. Primero que nada esta prueba permite evitar perder el tiempo buscando un algoritmo polinomial para resolver el problema: no encontrar no significa ser incompetente. Además, justifica buscar heurísticas, es decir, algoritmos que para la clase de problemas de optimización entreguen un valor “cercano” al óptimo.

7 Problemas NP-difíciles

Aquí el interés no es sólo en los problemas de reconocimiento como cuando se estudia las clases P , NP y NP completos. Un problema NP difícil es un problema al menos tan difícil como un problema NP-completo.

Proposición 7.1 *Sea O un problema de optimización. Si el problema de decisión asociado a O es NP-completo, entonces O es NP-difícil.*

Por ejemplo para el problema del vendedor viajero TSP es NP-difícil.