Problemas de decisión y búsqueda

Horst H. von Brand
vonbrand@inf.utfsm.cl

Departamento de Informática
Universidad Técnica Federico Santa María

Contenido

Decisión y búsqueda

Hasta ahora, nuestros problemas han sido *problemas de decisión*, se pregunta si hay o no algún objeto de interés. Eso en la práctica rara vez es suficiente: si el objeto existe, nos interesa saber cuál es; o sabemos que objetos del tipo buscado existen, queremos el mayor (o menor) de ellos. Son *problemas de búsqueda*.

Decisión y búsqueda

Por ejemplo, sabemos que un grafo conexo todos cuyos vértices son de grado par tiene un circuito de Euler (pasa una vez por cada arco). Lo anterior es fácil de verificar (*decisión*), pero no nos da luces sobre ningún circuito (*búsqueda*).

Decisión contra búsqueda para SAT

Supongamos que contamos con una función que decide SAT, es decir, dada $\phi(x_1, x_2, \ldots, x_n)$ la llamada SAT$(\phi(x_1, x_2, \ldots, x_n))$ dice si esa expresión puede satisfacerse.

Búsqueda para SAT

function SAT-search($\phi(x_1, \ldots, x_n)$)
    if SAT($\phi(x_1, \ldots, x_n)$) **then**
        for $i \leftarrow 1$ **to** n **do**
            $v_i \leftarrow \text{SAT}(\phi(v_1, \ldots, v_{i-1}, \text{T}, x_{i+1}, \ldots, x_n))$
        end
        return v
    else
        return $\perp$
    end
end

Autoreducción

A la situación descrita para SAT, de reducir la solución del problema de búsqueda a un número polinomial de soluciones de problemas de decisión (no mayores al problema original) le llamaremos *autoreducción*.

Búsqueda para SAT

El algoritmo dado es bastante obvio, llama a la función SAT $n+1$ veces.

Si la función SAT tomara tiempo polinomial en el largo de la expresión, SAT-search tomaría tiempo polinomial en hallar un vector de valores $\mathbf{v}$ que satisface ϕ.

Si la operación de búsqueda para SAT tomara tiempo polinomial en hallar una asignación de valores a los x_i que hace verdadera la expresión $\phi(x_1,\ldots,x_n)$, podríamos usarla para resolver el problema de decisión en tiempo polinomial.

Las complejidades de ambos problemas están relacionadas.

Oráculos

Una manera de formalizar lo anterior es suponer que tenemos una caja mágica $O_{\mathsf{SAT}}(\phi)$ (un *oráculo*) que resuelve una instancia de SAT en un paso, y la usamos como subrutina en nuestra búsqueda. Es claro que «solución en un paso» es completamente irreal, pero suponer cualquier otro costo de tiempo polinomial no hace diferencia en las conclusiones.

Búsqueda para Clique

El problema de decisión Clique toma un grafo G y un entero k y pregunta si G tiene una *clique* de tamaño k.

Un problema de búsqueda respectivo da $G = (V, E)$ y pide una *clique* de tamaño máximo.

Para simplificar notación, llamaremos $G \setminus v$ al grafo G sin el vértice v (ni los arcos en que participa, claro está).

Búsqueda para Clique

function Clique-search(G)
    $k \leftarrow |V|$
    while $\neg O_{\text{Clique}}(G, k)$ **do**
        $k \leftarrow k - 1$
    end
    foreach $v \in G$ **do**
        if $O_{\text{Clique}}(G \smallsetminus v, k)$ **then**
            $G \leftarrow G \smallsetminus v$
        end
    end
    return G
end

Búsqueda para Clique

Este algoritmo ejecuta O_{Clique} a lo más $2|V|$ veces. Esto claramente está acotado por un polinomio en el tamaño de la descripción de G.

Notación para tratar este tema

Un lenguaje L está en NP si hay una demostración sucinta de que $\sigma \in L$, el certificado c. Usamos máquinas de Turing deterministas para calcular funciones, la máquina f se inicia con una palabra (la codificación de varias si es una función de más de un argumento) en su cinta, lo que queda en la cinta cuando se detiene es el valor de f. Si no se detiene, diremos que el valor es $\perp$.

Notación para tratar este tema

Definición

Un *verificador* es una función ρ que toma dos argumentos σ y c, y retorna 0 (falso) o 1 (verdadero). Decimos que ρ es un NP-verificador si hay un polinomio p tal que para todo σ, c el cálculo de $\rho(\sigma, c)$ se detiene y responde en a lo más $p(|\sigma|)$ pasos.

Estamos suponiendo que el cálculo de ρ siempre se detiene, no permitimos $\perp$ como resultado. Note que definimos el costo de ρ solo en términos de su primer argumento. Pero si da a lo más $p(|\sigma|)$ pasos antes de responder, nunca puede revisar más de ese comienzo de c, y para σ dado el conjunto de certificados a considerar es efectivamente finito.

Notación para tratar este tema

Usamos verificadores para definir lenguajes.

Note que pueden haber varios certificados para un problema dado, por ejemplo, un grafo puede tener varias *clique* de tamaño k; si un grafo puede colorearse con tres colores, solo permutando los colores tiene al menos seis coloreos diferentes; pueden haber muchas asignaciones de valores a variables que hacen verdadera una expresión lógica.

También pueden haber certificados totalmente diferentes, como en el caso de circuitos eulerianos de grafos: un tipo de certificado es dar un circuito, otro diferente es listar las paridades de los grados de los vértices. Siempre podemos recurrir a los pasos que sigue una máquina de Turing no-determinista al reconocer la palabra.

Notación para tratar este tema

Definición

Sea ρ un verificador. El *conjunto testigo* de σ es:

$$\rho(\sigma) = \{c : \rho(\sigma, c) = 1\}$$

El *lenguaje definido por* ρ es:

$$L_\rho = \{\sigma : \rho(\sigma) \neq \varnothing\}$$

Note que estamos usando ρ en varios sentidos distintos.

Notación para tratar este tema

Es simple verificar lo siguiente, es solo malabarismo de definiciones.

Proposición

Un lenguaje L está en NP *si y solo si hay un* NP*-verificador ρ tal que $L = L_\rho$.*

Problemas asociados a un verificador

Asociado a un verificador ρ hay dos problemas con dato de entrada σ:

Decisión: ¿Hay algún c tal que $\rho(\sigma, c) = 1$?

Búsqueda: Retorne un c tal que $\rho(\sigma, c) = 1$, si existe; si no hay ninguno, retorne $\perp$.

Problemas asociados a un verificador

Definición

Sea ρ un verificador. Decimos que *búsqueda se reduce polinomialmente a decisión* para ρ si hay un algoritmo polinomial que dado σ y un oráculo para L_ρ entrega c tal que $\rho(\sigma, c) = 1$ si hay alguno, o $\perp$ en caso contrario.

En estos términos, para el verificador EVAL (dada la fórmula lógica ϕ y los valores de las variables —un certificado de que puede satisfacerse— retorna el valor de ϕ) búsqueda se reduce polinomialmente a decisión. Igualmente, para el verificador tras Clique búsqueda se reduce polinomialmente a decisión. (Claro que en rigor deberemos retornar solo un grafo de k vértices, no una *clique* mayor.)

¿Siempre puede reducirse búsqueda a decisión?

La respuesta pareciera ser obviamente afirmativa, en vista de NP-completitud: si $A \in \text{NP}$, sabemos $A \leq_p \text{SAT}$. Si podemos decidir SAT, podemos decidir A. Pero búsqueda se reduce polinomialmente a decisión para (el verificador EVAL detrás de) SAT, no para A. Contamos con un oráculo para A, no podemos aplicar el algoritmo de auto-reducción de SAT visto antes. Si A es NP-completo a su vez, las cosas sí funcionan.

Reducir búsqueda a decisión en NP-completos

Lema

Sea ρ un NP*-verificador, L_ρ* NP*-completo. Entonces hay funciones f, W computables en tiempo polinomial tales que para todo σ:*

1. *$\sigma \in L_\rho$ si y solo si $f(\sigma) \in$ SAT*
2. *Si T es una asignación de valores a $x_1, \ldots, x_n$ que satisfacen $\phi = f(\sigma)$, entonces $\rho(\sigma, W(\sigma, T)) = 1$*

La demostración es malabarismo con las definiciones. Estamos diciendo que si $L_\rho \in$ NP, podemos reducir $\sigma \in L_\rho$ polinomialmente a SAT. Un certificado c para σ también podemos traducirlo en tiempo polinomial en el correspondiente certificado T para esa instancia de SAT.

Reducir búsqueda a decisión en NP-completos

Teorema

Sea ρ un NP*-verificador. Si L_ρ es* NP*-completo, entonces búsqueda se reduce polinomialmente a decisión para ρ.*

Reducir búsqueda a decisión en NP-completos

Demostración.

Dados σ y un oráculo O_A para A, buscamos un testigo para σ, si existe. Sabemos que $A \leq_p \text{SAT}$, con lo que existen las funciones f, W del lema. Sea $\phi = f(\sigma)$, si encontramos una asignación T que satisface ϕ, vía $c = W(\sigma, T)$ tenemos lo buscado.
La pregunta es hallar T, pero no podemos usar la auto-reducción de SAT, tenemos solo un oráculo para A. Pero como a su vez A es NP-completo, sabemos que $\text{SAT} \leq_p A$. Sea g una reducción polinomial de SAT a A, podemos construir un oráculo para SAT:

$$O_{\text{SAT}}(\phi) = O_A(g(\phi))$$

(traduzca la instancia de SAT en una instancia de A, y aplique el oráculo respectivo a esta). □

Búsqueda y decisión en no NP-completos

Si A no es NP-completo, los problemas podrían ser de complejidad diferente. Por ejemplo, la decisión puede ser fácil (en P) pero la búsqueda difícil.

Las clases NP y coNP

Necesitamos alguna terminología adicional.

Definición

Un problema está en coNP ($A \in \text{coNP}$) si $\overline{A} \in \text{NP}$.

Las clases NP y coNP

Es claro que $\mathsf{P} \subseteq \mathsf{NP} \cap \mathsf{coNP}$ (si podemos determinar en tiempo polinomial si $\sigma \in A$, estamos determinando en tiempo polinomial si $\sigma \notin A$). El punto es que si $\mathsf{P} \neq \mathsf{NP}$, no queda claro si $\mathsf{NP} = \mathsf{coNP}$. Si A está en NP podemos verificar rápidamente mediante un algoritmo no determinista (dado el certificado del caso) que $\sigma \in A$, verificar que $\sigma \notin A$ no necesariamente es tan sencillo.

Por ejemplo, en el caso de SAT estamos preguntando si hay *alguna* asignación a las variables que hacen verdadera ϕ (con certificado obvio la asignación del caso), el complemento es determinar si *ninguna* asignación a las variables hace verdadera ϕ. Lo segundo parece más difícil (de partida, adivinar y verificar no se ve aplicable, parece que hay que revisar todas las asignaciones posibles).

Las clases NP y coNP

Se sospecha que si $P \neq \text{NP}$ entonces $\text{NP} \neq \text{coNP}$, y asimismo está abierto si $\text{P} = \text{NP} \cap \text{coNP}$.

Relación entre decisión y búsqueda

Teorema

Suponga que $\mathsf{P} \neq \mathsf{NP} \cap \mathsf{coNP}$. *Entonces hay un* NP*-verificador* ρ *tal que* $L_\rho \in \mathsf{P}$ *pero el problema de búsqueda respectivo no puede resolverse en tiempo polinomial (en particular, búsqueda no reduce polinomialmente a decisión).*

Relación entre decisión y búsqueda

Demostración.

Sea $A \in (\mathrm{NP} \cap \mathrm{coNP}) \setminus \mathrm{P}$. Entonces:

- $A = L_{\rho_1}$ para algún NP-verificador ρ_1, ya que $A \in \mathrm{NP}$.
- $\overline{A} = L_{\rho_2}$ para algún NP-verificador ρ_2, ya que $\overline{A} \in \mathrm{NP}$.

Para todo σ hay ya sea c_1 tal que $\rho_1(\sigma, c_1) = 1$ o c_2 tal que $\rho_2(\sigma, c_2) = 1$, pero no ambos. Definamos:

$$\rho(\sigma, c) = \rho_1(\sigma, c) + \rho_2(\sigma, c)$$

Para todo σ hay c tal que $\rho(\sigma, c) = 1$, o sea, $L_\rho = A \cup \overline{A} = \Sigma^*$. El problema de decisión para L_ρ es trivial, con lo que está en P.

Relación entre decisión y búsqueda

El problema de búsqueda para ρ no puede resolverse en tiempo polinomial, dada nuestra suposición que $A \notin \mathrm{P}$. Supongamos que dado σ podemos hallar c en tiempo polinomial tal que $\rho(\sigma, c) = 1$. Verifique (en tiempo polinomial) si $\rho_1(\sigma, c) = 1$. Hemos determinado en tiempo polinomial si $\sigma \in A$, cosa que supusimos imposible. □

Resumen

- Exploramos la relación entre problemas de decisión y de búsqueda.
- Si el problema está en NP, son «equivalentes» (pueden reducirse polinomialmente entre ellos); si no está en NP, no necesariamente es así.