Algoritmos pseudo-polinómicos

Marcos Herrero Agustín

¿Es polinómico?

```
vector<int> factorizar(int n) {
  vector<int> factores;
  int i = 2;

  while (i <= sqrt(n)) {
    while (n % i == 0) {
      factores.push_back(i);
      n /= i;
    }
    ++i;
  }

  if (n != 1)factores.push_back(n);
  return factores;
}
```

Coste $O(\sqrt{n})$

No es un algoritmo polinómico, porque el tamaño de la entrada no es n sino $\log n$.
Es lo que llamaremos un algoritmo pseudo-polinómico

Contenidos

El tamaño de la entrada

El tamaño de la entrada

- Toda instancia de un problema abstracto ha de representarse como una palabra en términos de un cierto alfabeto para poder ser procesada por una máquina de Turing. Llamamos sistema de representación a este alfabeto y tamaño de la instancia al número de símbolos del sistema utilizados para representarla
- A efectos prácticos, no importa el tamaño exacto. Pero es relevante si la diferencia en el número de símbolos empleados en dos sistemas de representación crece exponencialmente con el tamaño de la instancia, ya que esto desplaza la frontera entre lo que se considera tratable(polinómico) y lo que no. Por ejemplo, para representar un número natural n el número de símbolos utilizado es:
  - En base 2: $[\log_2 n] + 1 \sim \log n$
  - En base 10: $[\log_{10} n] + 1 \sim \log n$
  - En "base 1": $n \nsim \log n$

Para evitar estas diferencias al medir el tamaño de un problema, nos restringiremos a aquellos sistemas razonables y concisos, en el sentido de ser lo bastante similares al sistema binario.

- Decimos que un sistema de representación es razonable si toda instancia codificada en él puede ser transformada a sistema binario en un tiempo polinómico en el número de símbolos.
- Decimos que un sistema de representación es conciso si el número de símbolos utilizados para representar una instancia está en el orden del utilizado por un computador. Por tanto, el sistema unario no se considera conciso

Independencia de la representación

Nos gustaría hacer las nociones que vamos a tratar independientes del sistema de representación. Esto se consigue asumiendo que todos problema abstracto consta de dos funciones Length y Max definidas como siguen.

- Length asigna a cada instancia del problema el número de símbolos necesarios para representarla en alguna codificación razonable y concisa. Se corresponde con lo que hemos llamado tamaño del problema
- Max asigna a cada instancia el mayor (en valor absoluto) parámetro entero de esta (números no enteros se consideran composición de varios valores enteros). La llamaremos magnitud del problema. Si las instancias de un problema no tienen ningún parámetro numérico, establecemos por convenio que Max $\equiv 1$ para este problema

Nuevamente, no importan los valores exactos, sino el orden de los mismos.

Ejemplos

- SAT-FNC: recibe conjunto $C = \{c_1, ..., c_m\}$ de cláusulas
  $\text{Length}(I) = \sum_{i=1}^{m} |c_i|$, $\text{Max}(I) = 1$
- Cliqué: recibe un grafo $G = (V, E)$ y un entero $k \leq |V|$
  $\text{Length}(I) = |V| + |E|$, $\text{Max}(I) = k$
- Partición: recibe un conjunto de enteros $A = \{a_1, ..., a_n\}$
  $\text{Length}(I) = |A| + \sum_{i=1}^{n} \log a_i$, $\text{Max}(I) = \max_{1 \leq i \leq n} a_i$
- Mochila: recibe un conjunto U de pares peso-valor $(p(u), s(u))$ y dos enteros W(capacidad) y k(valor a alcanzar)
  $\text{Length}(I) = |U| + \log(W) + \log(k)$, $\text{Max}(I) = W + k$
- Viajante: recibe un conjunto $C = \{c_1, ..., c_m\}$ de ciudades, las distancias entre cada par de ellas $d(c_i, c_j), i, j \in 1, ..., m$ y un entero k (máxima distancia)
  $\text{Length}(I) = |C| + \sum_{i<j} \log d(c_i, c_j,)$, $\text{Max}(I) = \max_{i<j} d(c_i, c_j) + k$

Algoritmos pseudopolinómicos

Algoritmos pseudo-polinómicos

- Recuérdese que un algoritmo es polinómico si tiene coste polinómico en el tamaño del problema, es decir, coste acotado por un polinomio en la variable Length(I)
- De forma análoga, se dice que un algoritmo es pseudo-polinómico si tiene coste acotado por un polinomio en las variables Length(I) y Max(I). Evidentemente, todo algoritmo polinómico es también pseudo-polinómico.

Observación

Un problema que solo presente algoritmos pseudo-polinómicos no polinómicos entra en la categoría teórica de intratable. Sin embargo, los algoritmos pseudo-polinómicos solo presentan comportamiento exponencial al tratar instancias que contienen números muy grandes, así que en la práctica algoritmo pseudo-polinómico puede ser tan útil como uno polinómico

Ejemplos de algoritmos pseudo-polinómicos: Mochila

```
int Mochila(vector<int> & pesos, vector<int> & valores, int W) {
    int n = pesos.size();

    vector<int> maxValor(W+1, 0);

    for (int i = 1; i <= n; ++i) {
        for (int j = W; j >= pesos[i-1]; --j) {
         maxValor[j] = max(valores[i - 1] +
             maxValor[j - pesos[i - 1]], maxValor[j]);
      }
    }

return maxValor[n][W];
}
```

Coste $O(nW)$

Problemas numéricos

Decimos que un problema es numérico si no existe un polinomio p tal que para toda instancia I del problema se verifique $\text{Max}(I) \leq p(\text{Length}(I))$

Observación

No todo problema con algún parámetro numérico se considera numérico según esta definición. Por ejemplo, el problema de decisión del Cliqué no es numérico, pues su magnitud k está acotado por su tamaño $|V|$

En los problemas no numéricos, la magnitud está acotada por una función polinómica del tamaño del problema, así que un algoritmo que lo resuelve es pseudopolinómico si y solo si es polinómico.

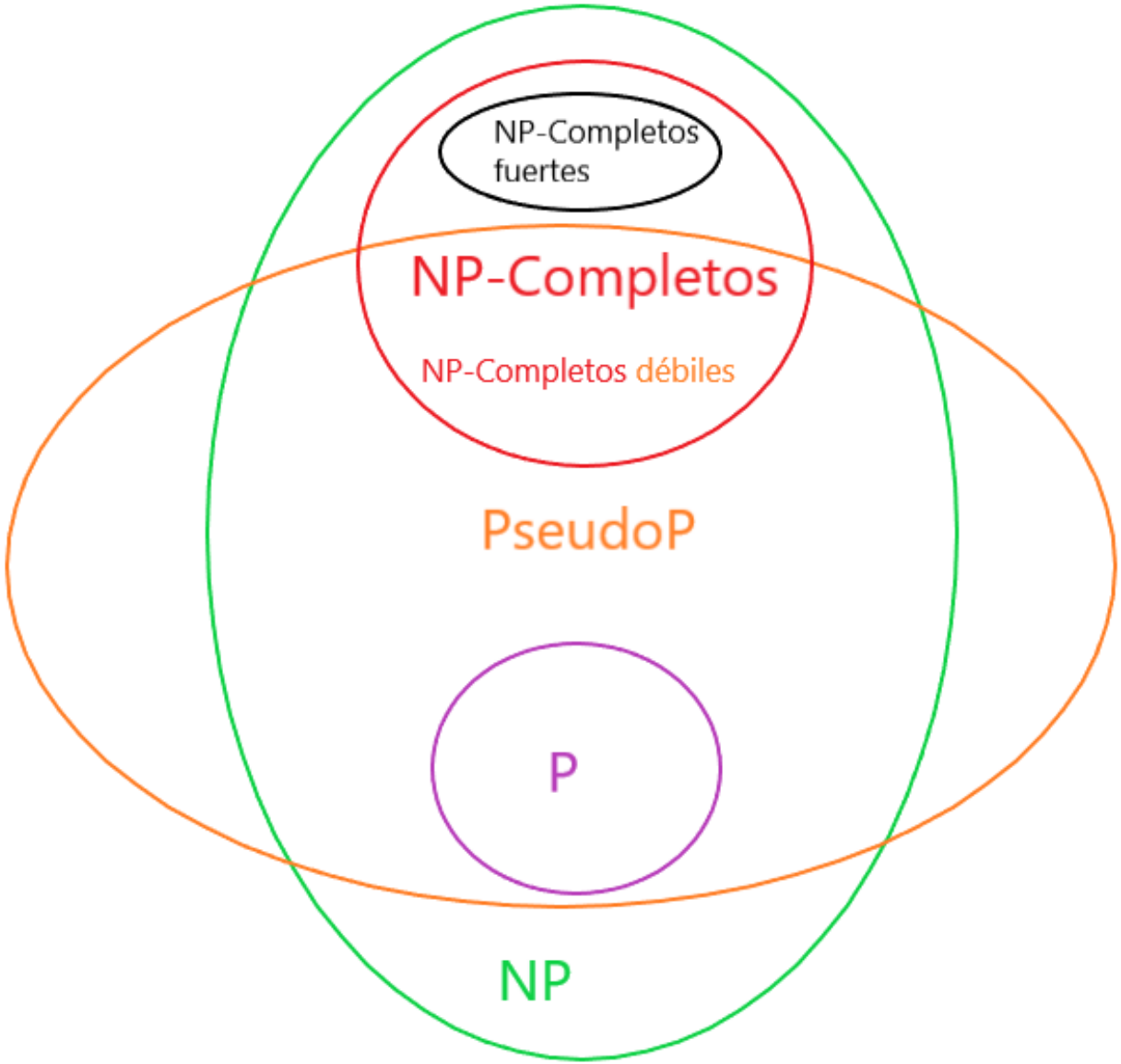
Clases de complejidad

NP-completos débiles y fuertes

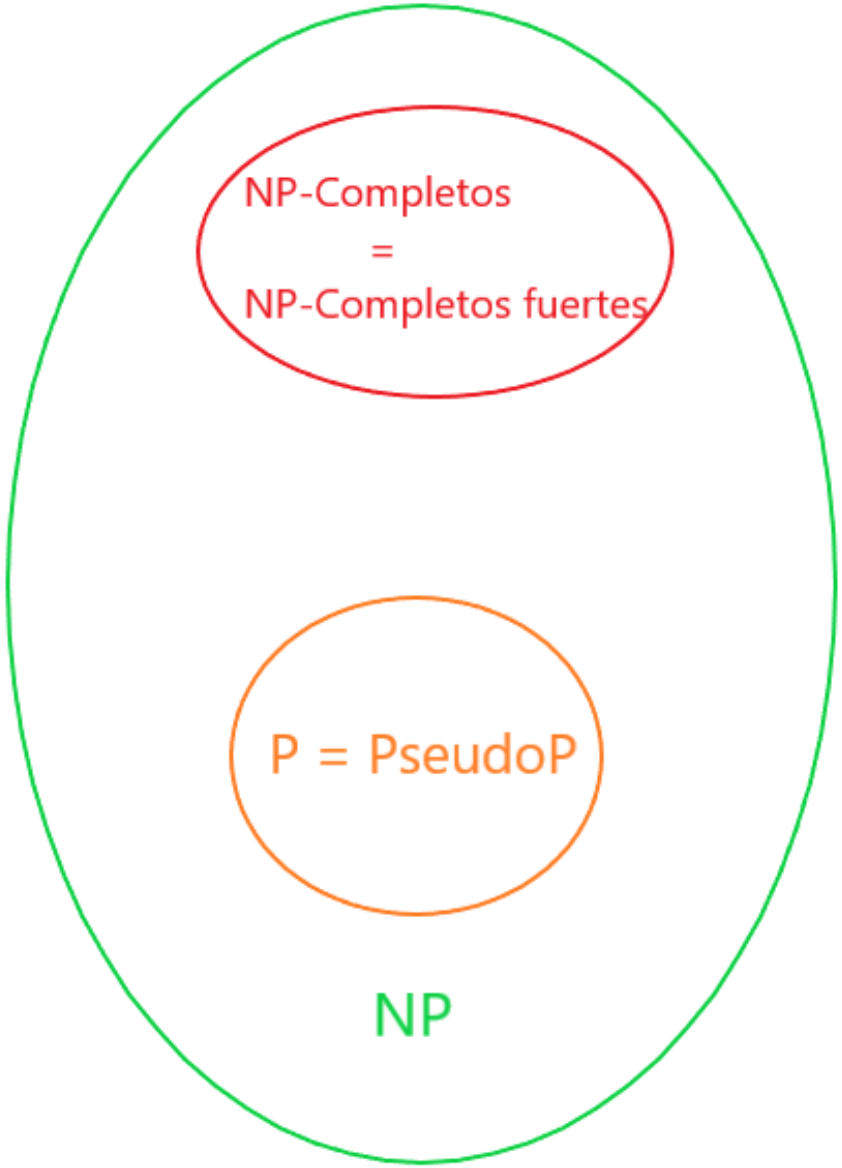
- Un problema NP-completo se dice NP-completo débil si existe un algoritmo pseudo-polinómico que lo resuelve. Por ejemplo: Mochila, Partición, Suma de subconjuntos,...
- Un problema NP-completo se dice NP-completo fuerte si existe un polinomio p tal que restringiendo el problema a aquellas instancias I con Max(I) acotada por p(Length(I)) sigue siendo NP-completo (es decir, seguiría siendo NP-completo aunque los datos se representaran en unario)

Un algoritmo pseudo-polinómico para un problema constituye un algoritmo polinómico para las instancias I de ese problema tales que Max(I) $\leq p$(Length(I)). Por tanto, si P$\neq$NP, las clases anteriores de problemas son disjuntas. No obstante, es una pregunta abierta si existen problemas NP-completos ni fuertes ni débiles.

Nuevas clases de complejidad



Nuevas clases de complejidad: problemas no numéricos



Ejemplos de problemas NP-Completos fuertes

- Todos los NP-Completos no numéricos: SAT, Cliqué, ciclo hamiltoniano, ...
- Los NP-Completos numéricos cuya magnitud se puede acotar por una constante y siguen siendo NP-Completos. Por ejemplo:
  - El problema del viajante
  - El problema del corte máximo
- Otros ejemplos:
  - 3-Partición
  - Bin Packing

El problema del viajante

Instancia: conjunto $C = \{c_1, ..., c_m\}$ de ciudades, distancias $d(c_i, c_j)$ entre cada par de ellas y un entero k
Pregunta: ¿Existe algún circuito que pase por cada ciudad exactamente una vez y cuya longitud sea menor o igual que k?

restringiendo las distancias a valores 0 y 1 y k a ser 0 el problema se puede interpretar como:

¿?

Instancia: conjunto $C = \{c_1, ..., c_m\}$ de ciudades y conjunto de A de aristas entre ellas
Pregunta: ¿Existe algún circuito formado por aristas de A que pase por cada ciudad exactamente una vez?

que es el problema del ciclo hamiltoniano

¿Cómo probar que un problema es NP-Completo fuerte?

- Encontrar un polinomio p tal que restringiendo el problema a instancias con magnitud $Max(I)$ acotada por $p(Length(I))$ sigue siendo NP-Completo
- Construir una reducción pseudo-polinómica de un problema que ya sabemos que es NP-completo fuerte.

Reducción pseudo-polinómica

Una transformación f de instancias de un problema A en instancias de un problema B es una reducción pseudo-polinómica de A a B si verifica:

a) Para toda instancia I de A, la respuesta de A para I coincide con la de B para $f(I)$

b) f es polinómica en las variables $\text{Length}_A(I)$ y $\text{Max}_A(I)$.

c) Existe un polinomio p tal que para toda instancia I de A: $p(\text{Length}_B[f(I)]) \geq \text{Length}_A(I)$

d) Existe un polinomio q tal que para toda instancia I de A: $\text{Max}_B[f(I)] \leq q(\text{Length}_A(I), \text{Max}_A(I))$

Consecuencias:

- Si B está en PseudoP y $A \leq_{pseudop} B$ entonces A está en PseudoP
- Si A es NP-completo fuerte y $A \leq_{pseudop} B$, entonces B es NP-completo fuerte

Relación con los algoritmos de aproximación

Para problemas NP-difíciles

Generalizamos la noción de NP-completitud fuerte a problemas de optimización:

- Un problema NP-difícil se dice NP-difícil débil si existe un algoritmo pseudo-polinómico que lo resuelve. Por ejemplo: Mochila, Partición, Suma de subconjuntos,...
- Un problema NP-difícil se dice NP-difícil fuerte si existe un polinomio p tal que restringiendo el problema a aquellas instancias con magnitud Max(I) acotada por p(Length(I)) sigue siendo NP-difícil (es decir, seguiría siendo NP-difícil aunque los datos se representaran en unario)

Nos gustaría ser capaces de encontrar algoritmos de aproximación precisos y eficientes para estos problemas

Esquemas de aproximación

- Un esquema de aproximación (AS) de un problema de optimización es un algoritmo que toma como entrada una instancia del problema y $\varepsilon \in \mathbb{R}^+$ y encuentra una solución aproximada del problema con error relativo menor o igual que ε. Es decir, si la solución óptima del problema para una instancia I es I^*, el algoritmo encuentra un valor $A(I)$ tal que $|A(I) - I^*| < \epsilon |I^*|$
- Nos interesan aquellos que tienen coste polinómico:
  - Los esquemas de aproximación en tiempo polinómico(PTAS) son polinómicos en Length(I)
  - Los esquemas de aproximación en tiempo totalmente polinómico (FPTAS) son polinómicos en Length(I) y $1/\varepsilon$
- Existe una relación estrecha entre la existencia de esquemas de aproximación polinómicos y la de algoritmos pseudopolinómicos

Pseudopolinómico $\Longrightarrow$ PTAS

- Existen técnicas que permiten transformar algoritmos pseudopolinómicos exactos para problemas de optimización en PTAS para el mismo problema.
- Para ciertos problemas estas técnicas funcionan muy bien: por ejemplo, por estos medios se ha obtenido un FPTAS del problema de la Mochila
- Inconvenientes:
  - No sirven con todos los algoritmos pseudopolinómicos
  - En muchos casos se obtienen PTAS exponenciales en $1/\varepsilon$, que resultan menos útiles. En particular, el problema de la mochila con dos restricciones(peso + volumen) presenta un algoritmo pseudopolinómico y un PTAS derivado de él, pero se puede probar que no admite ningún FPTAS
  - Los PTAS obtenidos, aunque sean eficientes en tiempo, suelen requerir coste elevado en espacio, lo que los hace impracticables para valores pequeños de ε

FPTAS $\Longrightarrow$ Pseudopolinómicos

Se puede probar que si un problema de optimización admite un FPTAS, bajo condiciones muy generales, admite un algoritmo pseudopolinómico. En concreto:

Teorema

Sea P un problema de optimización tal que el valor de toda solución factible suya es entero positivo y el valor óptimo de la función objetivo está acotado superiormente por un polinomio en las variables Length(I) y Max(I). Si P tiene un FPTAS, entonces tiene un algoritmo polinómico que lo resuelve

Ya hemos visto que el recíproco no es cierto (mochila con dos restricciones)
En consecuencia, salvo si P=NP, la gran mayoría de los problemas NP-difíciles fuertes no se pueden aproximar mediante FPTAS

Conclusiones

- Un algoritmo pseudo-polinómico puede resultar útil porque:
  - Solo presenta comportamiento exponencial al tratar números muy grandes, no necesarios en muchas aplicaciones prácticas
  - Frecuentemente, existen métodos para transformarlo en un PTAS del mismo problema
- Determinar si un problema es NP-difícil fuerte es útil en la práctica porque, si la respuesta es afirmativa, se tendrá que el problema no puede tener ningún algoritmo pseudo-polinómico y, bajo condiciones muy generales, tampoco ningún FPTAS

Bibliografía

- M. Garey y D.Johnson *Computers and Intractability: A Guide to the theory of NP-completeness*
- R. Neapolitan *Foundations of Algorithms*
  Capítulo 9
- M. Garey y D.Johnson *Strong NP-Completeness Results: Motivation, examples and implications*
- P.Schuurman y G.Woeginger *Aproximation Schemes - A tutorial*