

2) Usar  $\Theta(D)$ 

Idea: usamos un arreglo  $B$  de  $D$  bits, tal que

$B[i] = 1 \iff V[i]$  ha sido modificado.

Claramente nos encontraremos con el problema de inicializar  $B$ .

Para ello, consideremos  $B$  como un arreglo  $B'[1, D']$

con  $D' = \frac{D}{w} \leftarrow$  tamaño de palabra del procesador

Supongamos  $w \geq \log D$

Usamos la técnica anterior para inicializar  $B'$  en  $\Theta(D')$

$\Rightarrow$  esto usa  $2D' \log D' = \frac{2D}{w} \log D' \leq \frac{2D}{\log D} \log D \leq 2D$  bits

Junto a  $B$ , esto usa  $3D$  bits ( $\in \Theta(D)$ )

• Branch & Bound



## "Búsqueda exhaustiva inteligente"

### 1) Backtracking

• Mirando parte de una solución puedo descartar gran parte del espacio de soluciones.

Ej: si tengo  $(x_1, v, x_1, v, x_1)$ , ya que sé que  $x_1 = F$  no lleva a nada.

Para Backtracking necesitamos un "test" que mire un subproblema y diga:

- ¿Solución
- Encontré una solución
- No sé

acotar su costo

Luego el backtracking se ve algo así:

- Dado un problema  $P_0$
- $S \leftarrow \{P_0\}$  es el conjunto de subproblemas activos.
- Mientras  $S$  no esté vacío:
  - escoger un subproblema  $P \in S$  y quitarlo de  $S$ .
  - expandir  $P$  en  $P_1, \dots, P_k$  subproblemas menores.
  - para cada  $P_i$ :
    - Si test ( $P_i$ ):
      - encuentra sol global  $\Rightarrow$  lo retorno
      - no encuentra solución  $\Rightarrow$  descarto  $P_i$
      - else se agrega  $P_i$  a  $S$ .
- Se anuncia "No solución"

2) Branch & Bound  $\Rightarrow$  para optimización (s.p.g. minimización)  
• hirando parte de una solución puedo acotar su costo

Aquí necesitamos una función lower-bound que acote por abajo el costo de cualquier solución completa que parta de una solución incompleta dada.

pendido viajero

ej: TSP. Si tenemos un tour parcial que pasa por un cpto  $S$  de nodos, completarlo necesita un camino en  $(V-S)$  y arcos de  $(V-S)$  a  $a$  y  $b$  (el inicio y fin del tour)

Luego Branch & Bound se ve algo así:

- Dado un problema  $P_0$
- $best \leftarrow \infty$
- $S \leftarrow \{P_0\}$  es el conjunto de subproblemas activos
- Mientras  $S$  no esté vacío:
  - escoger un subproblema  $P \in S$  y quitarlo de  $S$ .
  - expandir  $P$  en  $P_1, \dots, P_k$  soluciones parciales
  - para cada  $P_i$ :
    - $\rightarrow$  siguiente

## ESQUEMA DE APROXIMACION

- Para cada  $P_i$ :

\* Si  $P_i$  es una sol. completa  $\Rightarrow$  actualizar best.

\* Si no,

si  $\text{lower-bound}(P_i) < \text{best.}$

agregar  $P_i$  a  $S$

- Se retorna best.

$\Rightarrow$  Costo de completar  $S$   $\gg$  arco más liviano de  $a$  a  $(V-S)$   
+ arco más liviano de  $b$  a  $(V-S)$   
+ pesos del mínimo subgrafo  
que conecta todos los nodos de  $(V-S)$   
MST



2015年12月1日 (火)

# ESQUEMA DE APROXIMACIÓN $(1+\epsilon)$

PROBLEMA DE LA MOCHILA NP-COMPLETO, se deja aproximar (KNAPSACK subset-sum)

Versión de decisión:  $x_1, x_2, \dots, x_n > 0$  bits para repr  $t$   
tope  $t$ .  $|\text{input}| = \Theta(n \cdot \log t)$  puede ser 32 bits

Se pide encontrar un conjunto que sume exactamente  $t$ .

Versión de opt:  $t$  es máximo

Forma Greedy: recibo elemento y veo qué pasa si lo meto o no a la bolsa.

Nota:  $\Theta(n \cdot t)$  prog. dinámica, pres  $t$  es gigantesco,  
 $\hookrightarrow 2^{32}$ ?!  
NO es polinomial en el  
input  $\Theta(n \log t)$ ... es exponencial!

Veamos algoritmo exacto:

Exacto:

$L \leftarrow \langle 0 \rangle$  (lista ordenada crecientemente siempre)

for  $i \leftarrow 1$  to  $n$  (lo incluyo o no?)

a cada elemento de  $L$ , sumar  $x_i$   
 $\hookrightarrow L \leftarrow \text{merge}(L, L + x_i)$  (unión de todos los subconjuntos posibles)  
truncar  $(L, t)$  (eliminar subconjuntos que se pasan de  $t$ )  
return max  $L$  (máx  $t$ )

$x = 1, 3, 2$

$L \leftarrow \langle 0 \rangle$   $L + 1 = \langle 1 \rangle$

$L \leftarrow \langle 0, 1 \rangle$   $L + 3 = \langle 3, 4 \rangle$

$L \leftarrow \langle 0, 1, 3, 4 \rangle$   $L + 2 = \langle 2, 3, 5, 6 \rangle$

$L \leftarrow \langle 0, 1, 2, 3, 4, 5, 6 \rangle$

$= \langle 0, 1, 3, 4 \rangle$  agrego

$\langle 2, 3, 5, 6 \rangle$  no agrego

en el merge se eliminan repetidas

Lo anterior en  
Fuerza Bruta...

$$2^3 + 2^{n-1} + 2^{n-2}$$



3>

¿Cuánto tiempo requiere?  $\sim \Theta(2^n)$  pueden estar todos los subconjuntos posibles en L  
INACEPTABLE

Vamos a aproximar. La idea es hacer que L no se agrande mucho

- Versión aproximada: Dado un elemento  $z \in L$ , tendremos una  $L'$  donde garantizamos que un  $y \in L'$  que **representa** a  $z$  está en  $L'$ . Y represente a  $z$  si i

$$\left| \frac{z}{1+\delta} \leq y \leq z \right|$$

$z$  es representado por alguien un poco **MENOR** (no puedo usar mayor porque me podría pasar de  $t$ )

Aprox ( $\epsilon$ )

$L \leftarrow \langle 0 \rangle$

for  $i \leftarrow 1$  to  $n$

$L \leftarrow \text{merge}(L, L + x_i)$

$\text{truncar}(L, t)$

**Filtrar**( $L, \delta$ )  $\leftarrow$  borro los eltos representados por otros

return max L

**Filtrar**( $L, \delta$ )

$L' \leftarrow \langle L[1] \rangle$  tomo el min; por definición nadie lo puede representar.

$\text{last} \leftarrow L[1]$

for  $i \leftarrow 2$  to  $|L|$

if  $L[i] > \text{last} \cdot (1+\delta)$

$L' \leftarrow L' \text{ concat } L[i]$

$\text{last} \leftarrow L[i]$

return  $L'$

Este error relacionado a  $1+\delta$  se va acumulando.

Este error relacionado a  $(1+\delta)$  se va acumulando.

Después de filtrar, tenemos:

$$L[1] \geq 1$$

$$L[i] \geq (1+\delta) \cdot L[i-1]$$

} estos consecutivos van creciendo consecutivamente

$$\text{si } l = |L|,$$

$$L(l) \geq (1+\delta)^{l-1}$$

pero solo trabajamos mientras  $L(l) \leq t$  (al truncar)

$$\frac{1}{\log(\cdot)} \cdot (1+\delta)^{l-1} \leq t \quad \left( \log(1+\delta) \cdot (l-1) \leq \log t \right)$$

$$l \leq \frac{\log t}{\log(1+\delta)} + 1$$

tenemos esta sup. para el largo!

Entonces en cada iteración del for pago  $\mathcal{O}\left(1 + \frac{\log t}{\log(1+\delta)}\right)$

$$\cancel{\mathcal{O}(2^n)} \rightarrow \mathcal{O}\left(\frac{n \log t}{\log(1+\delta)}\right)$$

y  $n \log t$  es el tamaño del input.

Pareciera ser lineal en el tamaño del input, pero debemos estudiar bien qué sorpresas salen del  $(1+\delta)$

Aunque sean números enteros, hay que dejar que  $\delta > 0 \in \mathbb{R}$  si no explota.

¿Cuál es la relación entre  $\varepsilon$  y  $\delta$ ? peor caso

$$\frac{z}{1+\delta} \leq y \leq z$$

↓ y pasa a 2ª etapa. se reemplaza luego con  $x$

$$\frac{z}{(1+\delta)^2} \leq \frac{y}{1+\delta} \leq x \leq y$$

↓ pasa a 3ª etapa y se reemplaza por  $w$

$$\frac{z}{(1+\delta)^3} \leq \frac{x}{1+\delta} \leq w \leq x$$

∴ después de  $n$  iteraciones, entrego  $z^*$  que representa a  $z$  óptimo

$$\frac{z}{(1+\delta)^n} \leq z^* \leq z$$



$x_1$  solución o  $x_1$  el máx de todos,  $\leq t$

Lo peor es que el primer elemento sea  $x_1 = t$ , luego viene  $x_2 = t/(1+\delta)$  y el alg. elimina  $x_1$ , luego llega  $x_3 = \frac{t}{(1+\delta)^2}$ , se elimina  $x_2$  y se reemplaza por  $x_3$ , y así.

$$\Rightarrow \text{Luego digo } \frac{z}{(1+\delta)^n} \leq z^* \leq z$$

$1+\epsilon$  aproximado.

$$\begin{aligned} 1+\epsilon &= (1+\delta)^n \\ \Rightarrow (1+\epsilon)^{1/n} &= 1+\delta \\ \Rightarrow \delta &= (1+\epsilon)^{1/n} - 1 \end{aligned}$$

Cormen demuestra que  $\delta = \epsilon/2n$  funciona

Si originalmente había mucha distancia entre los elementos, quizás con filtrar no elimino ninguno de la lista, pero este problema era fácil por sé! porque al sumar los costos se llega rápido a  $t$ . El problema es difícil cuando los costos entre ellos son muy parecidos.

Reemplazando  $\delta$ , el costo del apróx es

$$\Theta\left(\frac{n^2 \log t}{\log(1+\epsilon)}\right)$$

$$\text{usaremos } \frac{\epsilon}{1+\epsilon} \leq \ln(1+\epsilon) \leq \epsilon$$

$$\Rightarrow \Theta\left(\frac{\epsilon [n, 2]}{(1+\epsilon) n^2 \log t}\right) = \boxed{\Theta\left(\frac{n^2 \log t}{\epsilon}\right)}$$

está en función de  $\frac{1}{\epsilon}$ , que era lo que queríamos

El problema es polinomial en  $n$ , en  $\log t$  y en  $\frac{1}{\epsilon}$ .

# FIN

NOT