

# **PROBLEMA DE LA MOCHILA REAL**

Problema de la mochila pero en vez de aplicar vuelta atrás como en 2º aplicamos algoritmo voraz. Ahora, a diferencia de en segundo los objetos se pueden FRACTIONAR.



UNIVERSIDAD  
**COMPLUTENSE**  
MADRID

**ALBERTO VERDEJO**

# Problema de la mochila real

Objetivo : MAXIMIZAR el valor de los objetos que entran en la mochila!!!

- ▶ Cuando Alí-Babá consiguió entrar en la Cueva de los Cuarenta Ladrones, encontró allí objetos muy valiosos, de los que conocía muy bien el peso y valor.
- ▶ Solo podía llevar consigo aquellas riquezas que cupieran en su pequeña mochila, que soportaba un peso máximo conocido.
- ▶ Suponiendo que los objetos fueran fraccionables, ¿qué objetos debería elegir Alí-Babá para maximizar el valor total de su mochila?



# Problema de la mochila real

Todos los objetos tienen un peso y un valor >0.

- ▶ Hay  $n$  objetos, cada uno con un peso  $p_i > 0$  y un valor  $v_i > 0$ .
- ▶ La mochila soporta un peso total máximo  $M > 0$ .
  - La suma de los pesos de todos los objetos de la cueva excede a  $M$
- ▶ Y el problema consiste en maximizar

$$\sum_{i=1}^n x_i v_i$$

Maximizar el valor de los objetos que metamos en la mochila.

con la restricción

$$\sum_{i=1}^n x_i p_i \leq M,$$

Con la restricción de que, la suma de los pesos de los objetos que metamos en la mochila, no puede superar el peso máximo de la mochila.

donde  $x_i$  es la *fracción* de objeto  $i$  tomada,  $0 \leq x_i \leq 1$ .

No todos los objetos caben en la mochila.

# Ejemplo

La estrategia es llenar la mochila cogiendo los objetos más valiosos primero.

$$M = 100, n = 5$$

	1	2	3	4	5
$p_i$	10	20	30	40	50
$v_i$	20	30	66	40	60
	10	20	66	40	60

Pero esta estrategia no es óptima porque si cogemos el x3, x5 y x2 tenemos que el valor de los objetos que metemos es 156 que es mejor que 146

seleccionar	$x_i$				valor	
máx. $v_i$	0	0	1	0,5	1	146

Los 1 significa que los cojo enteros y el 0'5 es que cojo la mitad, la fraccion

Consideraríamos los objetos de mayor valor.

## Ejemplo

Si consideráramos primero los de menor peso, obtendríamos un mejor valor.

$$M = 100, n = 5$$

	1	2	3	4	5
$p_i$	10	20	30	40	50
$v_i$	20	30	66	40	60

Pero NO es una estrategia que funciona siempre tampoco...

seleccionar	$x_i$					valor
máx. $v_i$	0	0	1	0,5	1	146
mín. $p_i$	1	1	1	1	0	156

El problema es que un objeto que puede valer mucho también puede pesar mucho

O un objeto que vale poco puede pesar poco.

# Ejemplo

Lo que realmente necesitamos considerar es la densidad de valor.

$$M = 100, n = 5$$

	1	2	3	4	5
$p_i$	10	20	30	40	50
$v_i$	20	30	66	40	60
$\frac{v_i}{p_i}$	2	1,5	2,2	1	1,2

Cogemos los objetos con mayor densidad de valor.  
Esto si que saca los máximos.

seleccionar	$x_i$					valor
máx. $v_i$	0	0	1	0,5	1	146
mín. $p_i$	1	1	1	1	0	156

# Ejemplo

$M = 100, n = 5$

	1	2	3	4	5
$p_i$	10	20	30	40	50
$v_i$	20	30	66	40	60
$\frac{v_i}{p_i}$	2	1,5	2,2	1	1,2

$\geq$     $\geq$     $\leq$     $\geq$

seleccionar	$x_i$					valor
máx. $v_i$	0	0	1	0,5	1	146
mín. $p_i$	1	1	1	1	0	156
máx. $\frac{v_i}{p_i}$	1	1	1	0	0,8	164

$10 + 20 + 30 + 50 \cdot 0,8$

ESTA  
ESTRATEGIA  
FUNCIONA  
SIEMPRE, LO  
DEMUESTRA  
DESPUÉS.

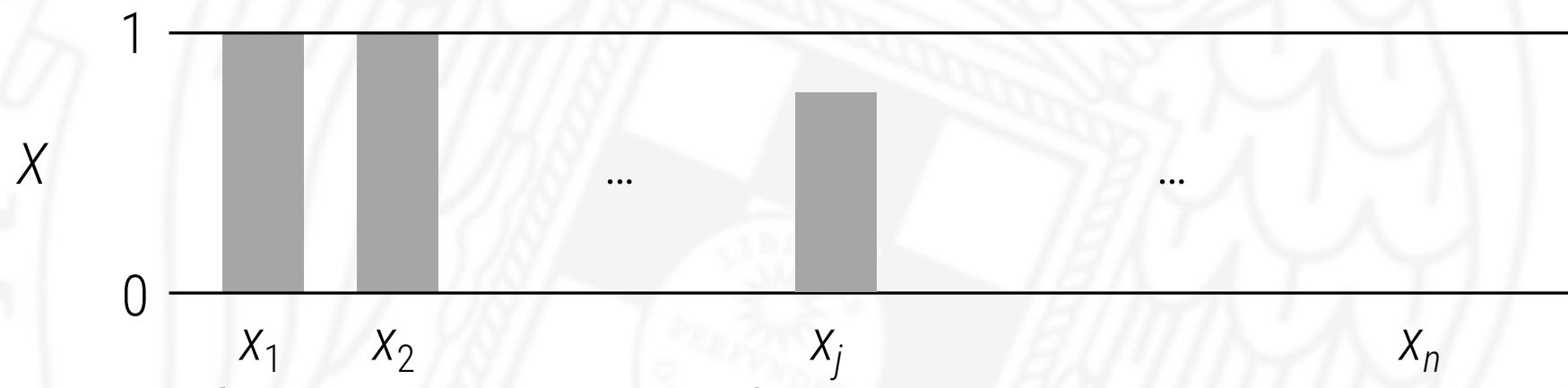
# Demostración de optimalidad

- ▶ Objetos ordenados de mayor a menor valor por unidad de peso:

$$\frac{v_1}{p_1} \geq \frac{v_2}{p_2} \geq \dots \geq \frac{v_n}{p_n}$$

Ordenados respecto a  $v_i/p_i$

- ▶ Solución  $X$  construida por el algoritmo voraz:

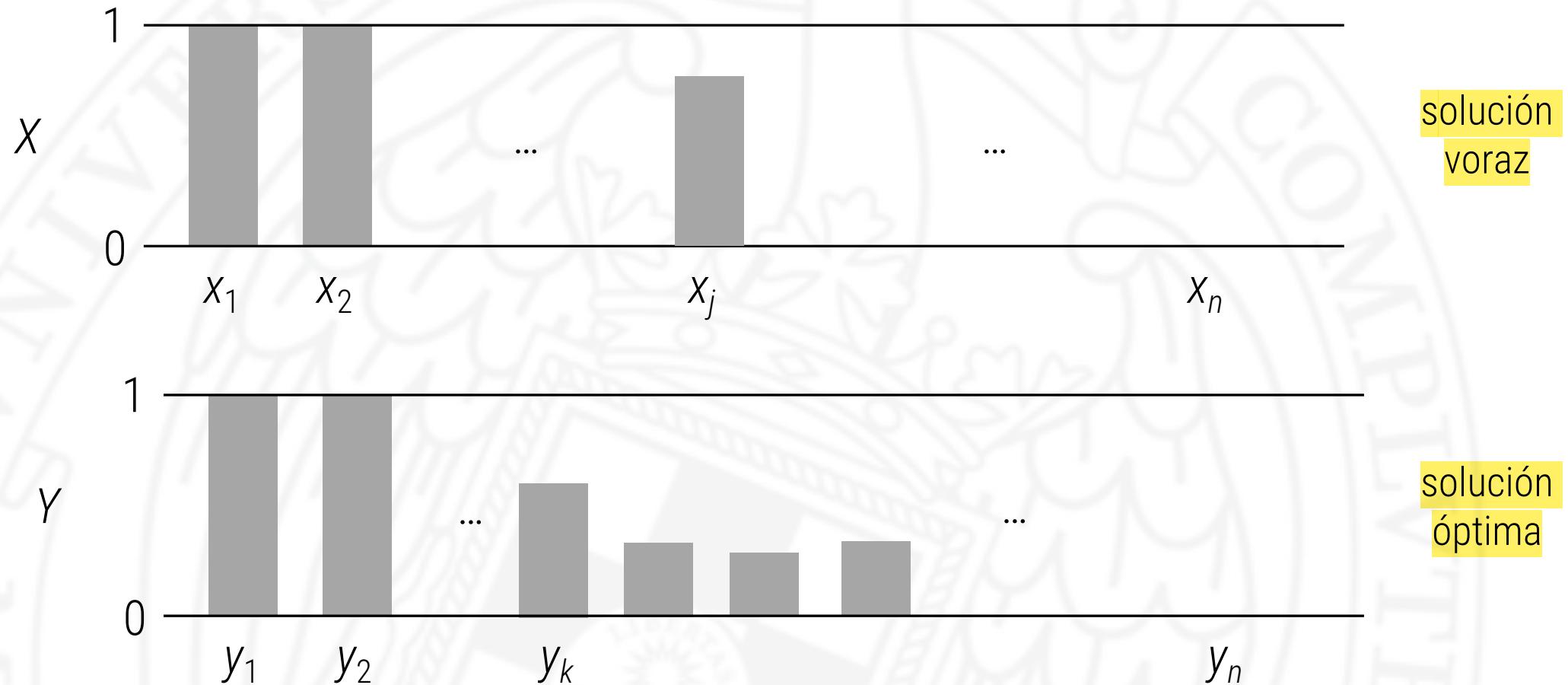


$x_j = 1$   
Objetos enteros

$0 \leq x_j < 1$   
objetos fraccionados

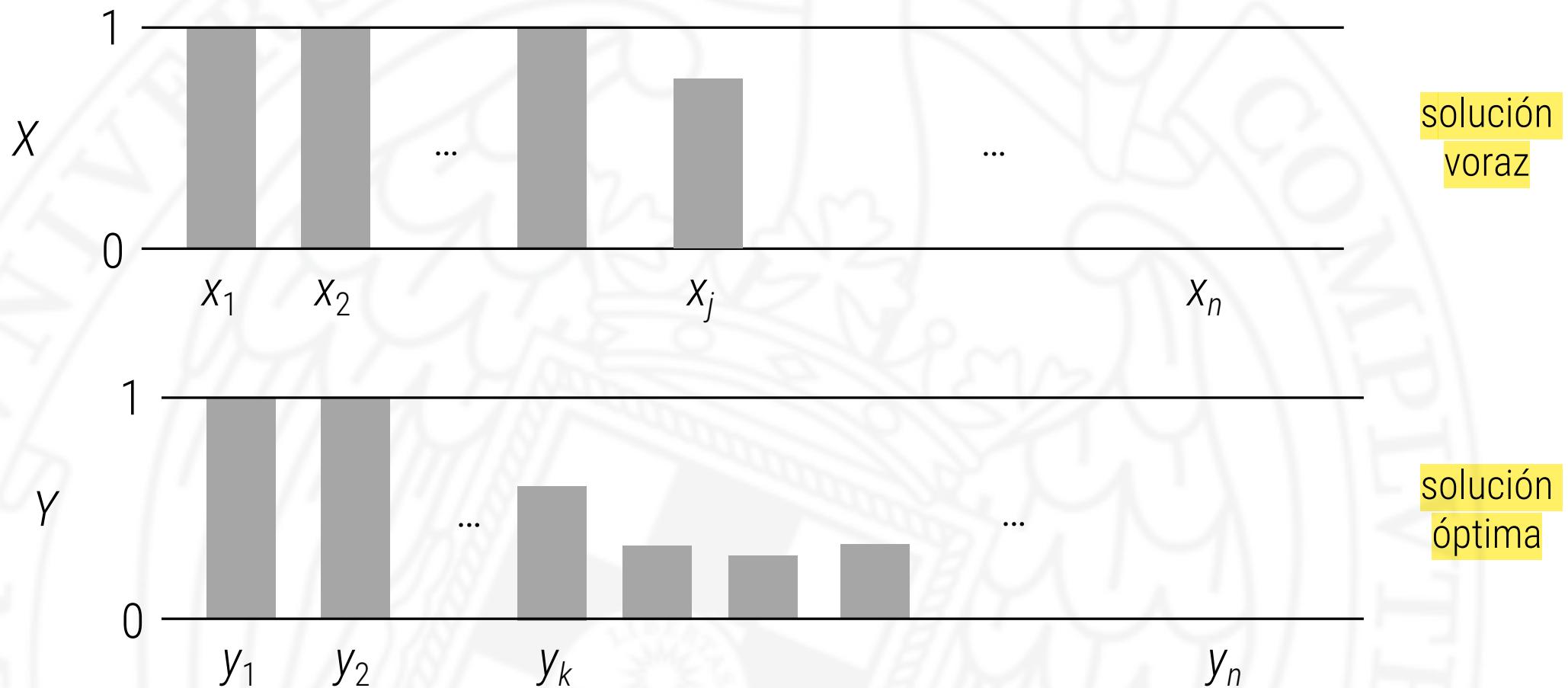
$x_j = 0$   
Objetos que no cogemos.

# Demostración de optimalidad



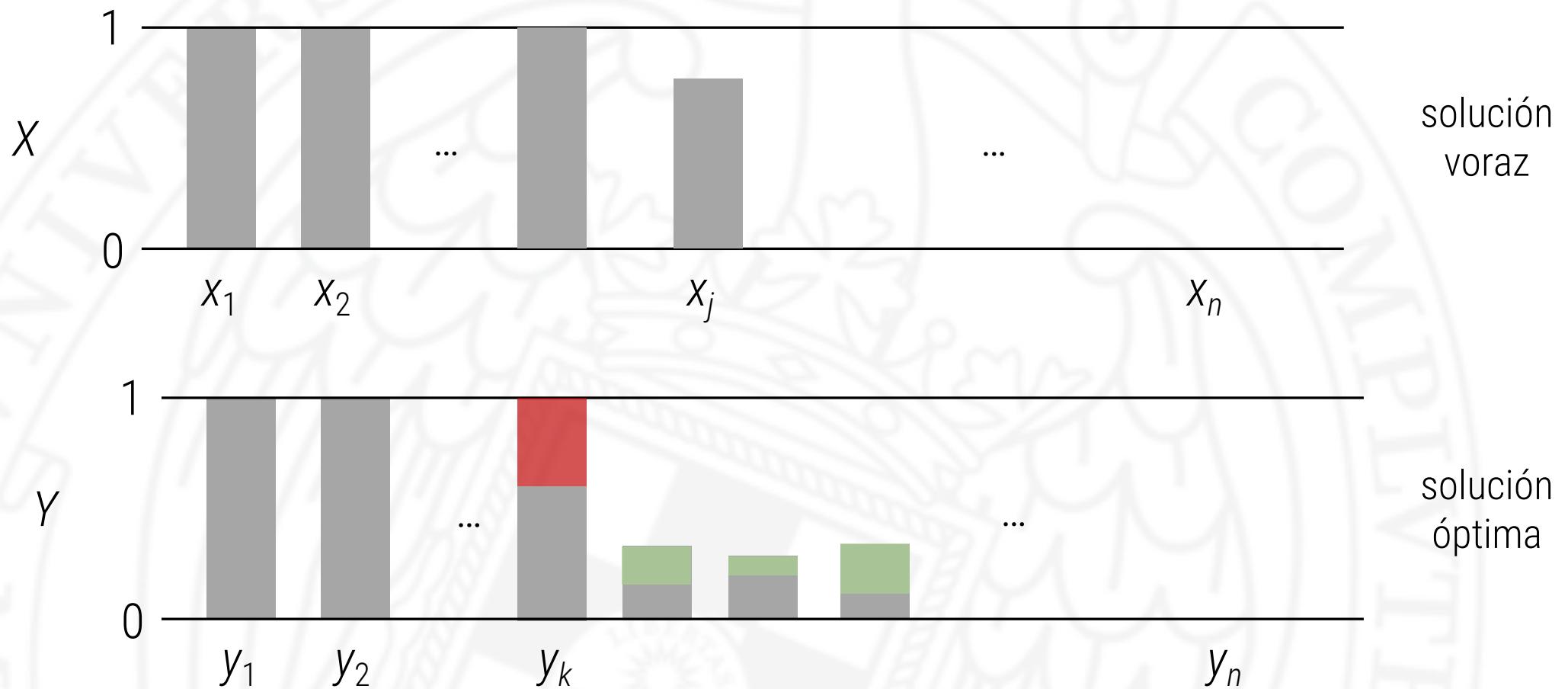
- ▶ Podemos asumir que  $\sum_{i=1}^n y_i p_i = M$ .

# Demostración de optimalidad



- ▶ Se debe cumplir que  $k \leq j$  y por tanto  $y_k < x_k$

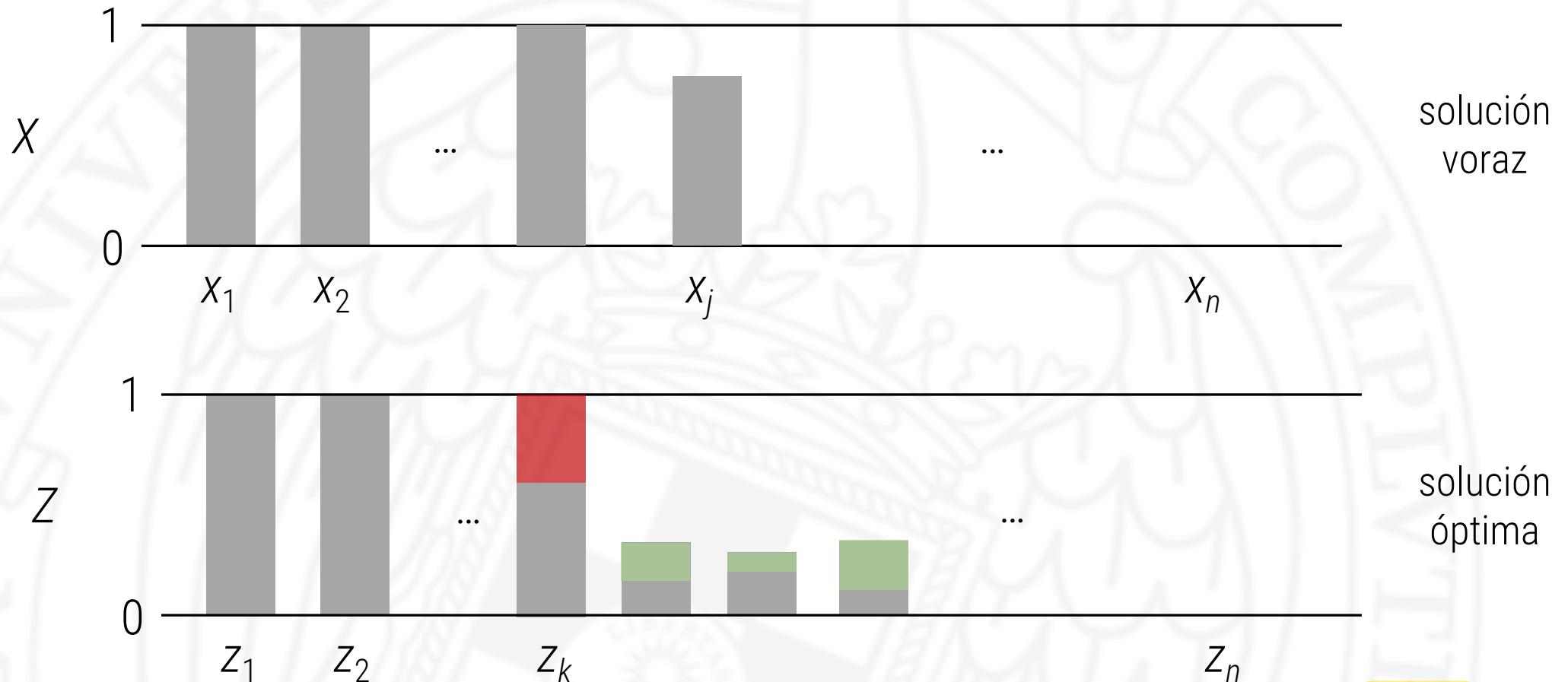
# Demostración de optimalidad



- Aumentar  $y_k$  para hacerlo igual a  $x_k$

$$\textcolor{red}{\square} = \textcolor{green}{\square} + \textcolor{green}{\square} + \textcolor{green}{\square}$$

# Demostración de optimalidad



- ▶ La nueva solución  $Z$  cumple

$$\sum_{i=k+1}^n p_i(y_i - z_i) = p_k(z_k - y_k)$$

El peso que quitamos para que sea igual que la solución voraz a de ser igual que el que metemos

# Demostración de optimalidad

►  $Z$  sigue siendo óptima:

$$\begin{aligned} \sum_{i=1}^n v_i z_i &= \sum_{i=1}^n v_i y_i + v_k(z_k - y_k) - \sum_{i=k+1}^n v_i(y_i - z_i) \\ &= \sum_{i=1}^n v_i y_i + \frac{v_k}{p_k} p_k(z_k - y_k) - \sum_{i=k+1}^n \frac{v_i}{p_i} p_i(y_i - z_i) \\ &\geq \sum_{i=1}^n v_i y_i + \underbrace{\left( p_k(z_k - y_k) - \sum_{i=k+1}^n p_i(y_i - z_i) \right)}_{0} \frac{v_k}{p_k} \\ &= \sum_{i=1}^n v_i y_i \end{aligned}$$

# Implementación

```
struct Objeto {  
    double peso;  
    double valor;  
};
```

```
struct Densidad {  
    double densidad; // valor / peso  
    int id; // identificador del objeto  
};
```

```
bool operator>(Densidad const& a, Densidad const& b) {  
    return a.densidad > b.densidad; Ordenamos de mayor a menor.  
}
```

Ordenar objetos por densidad de valor.

# Implementación

```
double mochila(vector<Objeto> const& objetos, double M,  
               vector<double> & sol) { // O(N log N)  
    int N = objetos.size();  
    vector<Densidad> D(N);  
    for (int i = 0; i < N; ++i) {  
        D[i].densidad = objetos[i].valor / objetos[i].peso;  
        D[i].id = i; // para saber a qué objeto corresponde  
    }  
    sort(D.begin(), D.end(), greater<Densidad>());
```

Primero calculamos las densidades y ordenamos el vector de mayor a menor.



Por eso el coste de la función es  $N \log N$

# Implementación

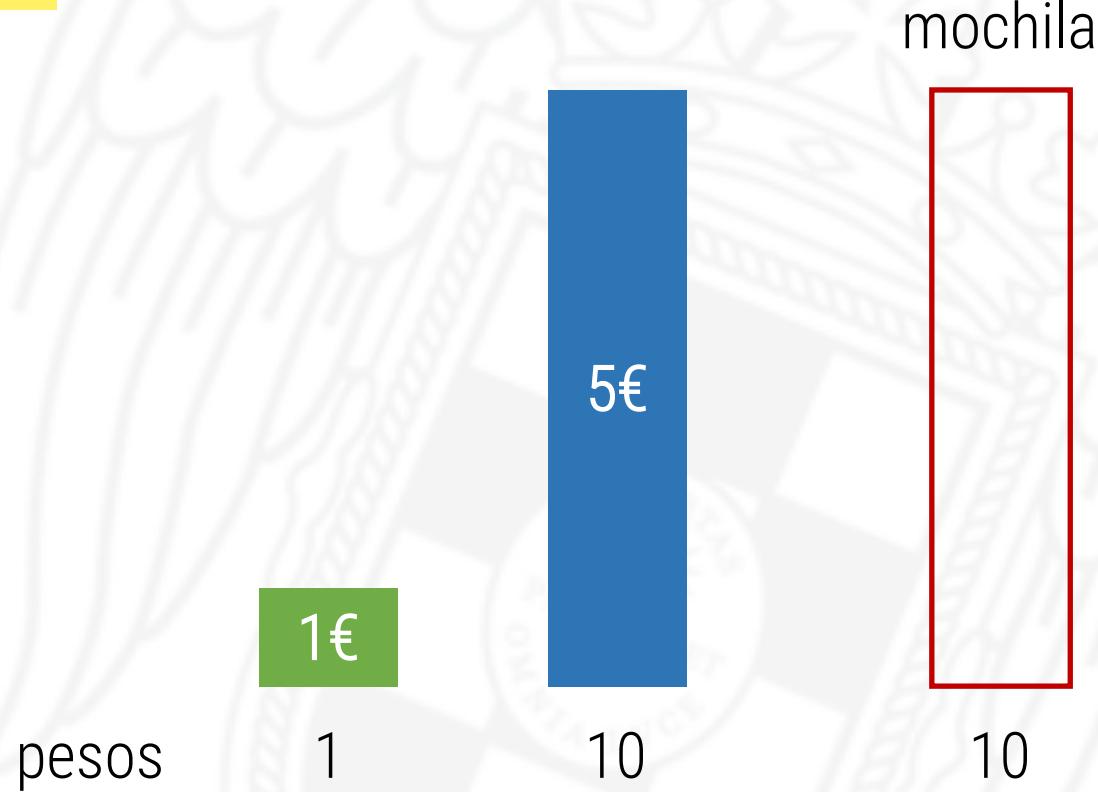
```
double peso = 0, valor = 0;  
int i;                                Mientras quepan enteros...  
for (i = 0; i < N && peso + objetos[D[i].id].peso <= M; ++i) {  
    sol[D[i].id] = 1; // el objeto D[i].id cabe completo  
    peso += objetos[D[i].id].peso;           Incrementamos el peso y el valor del  
    valor += objetos[D[i].id].valor;         contenido de la mochila.  
}  
  
if (i < N) { // partir el objeto D[i].id  
    sol[D[i].id] = (M - peso) / objetos[D[i].id].peso;  
    valor += sol[D[i].id] * objetos[D[i].id].valor;  
}  
  
return valor;  
}
```

Fracción del objeto  $i$  que cabe en la mochila para completarla.

$$\text{peso} + x_i \cdot p_i = M \Rightarrow x_i = \frac{M - \text{peso}}{p_i}$$

## Versión 0/1

- ▶ Los objetos no se pueden partir.
- ▶ La estrategia voraz anterior no siempre calcula una solución óptima.
- ▶ Contraejemplo:



Esta versión del problema se puede resolver mediante programación dinámica.