Problema de la Mochila (Knapsack Problem)

Formulación

- Se tiene una colección de objetos numerados i=1,..,n
- Cada objeto tiene un valor v_i>0 y un peso w_i>0.
- El objetivo es guardar en la mochila los objetos que sumen el máximo valor, sin sobrepasar el peso máximo permitido W.
- Sea $x_i \in [0,1]$ la fracción de cada objeto, entonces el problema es

$$\text{Maximizar} \quad \sum_{i=1}^{n} x_i v_i \qquad (1)$$

$$\text{Sujeto a} \quad \sum_{i=1}^{n} x_i w_i \leq W \qquad (2)$$

$$0 \leq x_i \leq 1$$

Componentes del problema

- Conjunto de candidatos: Los objetos $i \in \{1,..,n\}$
- El conjunto solución: La fracción seleccionada de cada objeto: Vector $(x_1,..,x_n)$
- Función de factibilidad: No superar peso máximo, ecuación (2). Los valores de $x_i \in [0,1]$.
- Función objetivo: Maximizar valor, ecuación (1)

Análisis preliminar

- Si la suma de los pesos de todos los objetos es menor a W, se pueden empacar todos y esto da el máximo valor: Caso trivial.
- Si la suma de los pesos es mayor a W, pero los objetos se pueden fraccionar, la solución óptima siempre llena la mochila a su máxima capacidad: Ecuación (2) se satisface con igualdad.

Algoritmo voraz genérico

```
función llenarMochila(v[1..n], w[1..n], W)
    candidatos←{1,..,n}
    para i←1 hasta n hacer x[i]←0
    peso←0
    mientras peso<W
        i ← seleccionarMejorObjetoRestante()    // A discutir
        candidatos←candidatos-{i}
        si peso+w[i]≤W
            x[i]←1
            peso←peso+w[i]
        sino
            x[i]←(W-peso)/w[i]
            peso←W
    devolver x
```

Qué función de selección usar?

Ejemplo:

Sea W=100 y la siguiente colección de objetos

	1	2	3	4	5
w	10	20	30	40	50
v	20	30	66	40	60
v/w	2.0	1.5	2.2	1.0	1.2

Seleccionando por orden de valor: {3, 5, 4}, V_{total}=146

Seleccionando por menor peso: {1, 2, 3, 4}, V_{total}=156

Seleccionando por razón v/w: {3, 1, 2, 5}, V_{total}=164

Algoritmo óptimo

Teorema: Si se seleccionan en orden decreciente de v/w, el algoritmo encuentra la solución óptima.

Dem:

Se numeran los objetos en el orden que son seleccionados: 1..n. Los objetos seleccionados son 1..j, de modo que $x_i=1$, $1 \leq i < j$; y $0 \leq x_i \leq 1$ para i=j.

Tenemos que para la solución óptima $\sum_{i=1}^{n} x_i w_i = W$

y para cualquier solución factible Y=(y_i) $\sum_{i=1}^{n} y_i w_i \leq W$

por lo tanto $\sum_{i=1}^{n} (x_i - y_i) w_i \geq 0$

Optimalidad

Los valores de la solución óptima $X=(x_i)$ y de una solución factible $Y=(y_i)$ son

$$V(X) = \sum x_i v_i \qquad\qquad V(Y) = \sum y_i v_i$$

y su diferencia es

$$\begin{aligned} V(X) - V(Y) &= \sum (x_i - y_i) v_i \\ &= \sum (x_i - y_i) w_i \frac{v_i}{w_i} \end{aligned}$$

se puede verificar además que para todo i, siendo j el punto de corte, se cumple

$$(x_i - y_i)\frac{v_i}{w_i} \geq (x_i - y_i)\frac{v_j}{w_j}$$

Prueba de optimalidad

y por lo tanto se concluye que

$$V(X) - V(Y) \geq \frac{v_j}{w_j} \sum (x_i - y_i) w_i \geq 0$$

es decir, no existe ninguna otra solución factible cuyo valor sea mayor que el valor de la solución dada por X.

Problema de la mochila con objetos no fraccionables

- Todo el análisis previo se hizo para el caso en que se puede tomar una fracción arbitraria de cada objeto entre 0..1.
- Esto aplica para cuando los objetos son “fluidos” ó las unidades en que se descomponen son tan pequeñas comparadas con el volumen total que efectivamente se aproximan muy bien como si fueran fluidos. Por ejemplo, bits en un sistema de almacenamiento con MBytes de capacidad.

Objetos no fraccionables

- En este caso el objeto se debe incluir en su totalidad, o no incluirlo en la lista de objetos seleccionados.
- Las variables x_i solo pueden tomar valores binarios {0,1}.
- La nueva formulación es

$$\text{Maximizar} \quad \sum_{i=1}^{n} x_i v_i \qquad (1)$$

$$\text{Sujeto a} \quad \sum_{i=1}^{n} x_i w_i \leq W \qquad (2)$$

$$x_i \in \{0, 1\}$$

Solución óptima

- El algoritmo voraz previamente visto no aplica en este caso. Contra-ejemplo.
- Es más, no se conoce una solución estrictamente polinómica para este problema.
  ➔ Pertenece a la clase de problemas ***NP-completos***.
- Algoritmo para solución óptima: Más adelante, programación dinámica.