PROBLEMA DE LA MOCHILA ENTERA (RAMIFICACIÓN Y PODA)

ALBERTO VERDEJO

UNIVERSIDAD COMPLUTENSE MADRID

Problema de la mochila (versión entera)

- Hay n objetos, cada uno con un peso $p_i > 0$ y un valor $v_i > 0$ (reales).
- La mochila soporta un peso total máximo $M > 0$.
- Y el problema consiste en maximizar

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

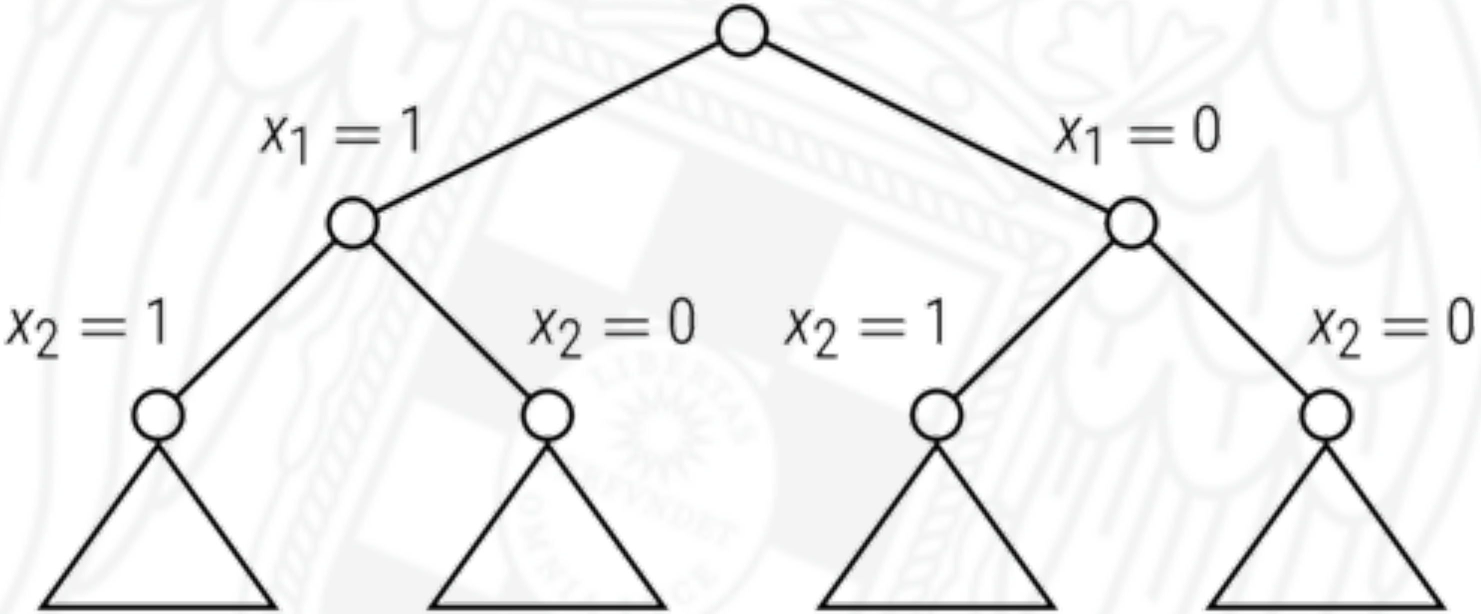
con la restricción

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

donde $x_i \in \{0, 1\}$ indica si hemos cogido (1) o no (0) el objeto i.

Representación de las soluciones y árbol de exploración

- Las soluciones son subconjuntos de objetos (no importa el orden), que representamos mediante tuplas $(x_1, \ldots, x_n)$, donde x_i indica si cogemos o no el objeto i.
- Árbol de exploración:



Estimación

- Como el problema es de maximización necesitamos una cota superior del beneficio de la mejor solución alcanzable desde un nodo:

1. Seleccionar todos los demás , $\sum_{i=1}^{k} x_i v_i + \sum_{i=k+1}^{n} v_i$
2. Utilizar el algoritmo voraz que resolvía este problema cuando los objetos se podían fraccionar $(0 \leq x_i \leq 1)$.

   Objetos ordenados en orden decreciente de valor por unidad de peso, v_i/p_i.

$$\underbrace{\textit{beneficio}}_{\text{acumulado}} + \underbrace{v_k + \text{estimación de } k+1 \text{ hasta } n}_{\text{estimación}}$$

Estimación

- Como el problema es de maximización necesitamos una cota superior del beneficio de la mejor solución alcanzable desde un nodo:

1. Seleccionar todos los demás , $\sum_{i=1}^{k} x_i v_i + \sum_{i=k+1}^{n} v_i$
2. Utilizar el algoritmo voraz que resolvía este problema cuando los objetos se podían fraccionar $(0 \leq x_i \leq 1)$.

   Objetos ordenados en orden decreciente de valor por unidad de peso, v_i/p_i.

$$\underbrace{\textit{beneficio} + v_k}_{\text{acumulado}} + \underbrace{\text{estimación de } k\text{+1 hasta } n}_{\text{estimación}}$$

Implementación

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

struct Nodo {
   vector<bool> sol;
   int k;
   double peso;            // peso acumulado
   double beneficio;       // valor acumulado
   double beneficio_est; // prioridad
   bool operator<(Nodo const& otro) const {
      return otro.beneficio_est > beneficio_est;
   }
};
```

Implementación

```
// estrategia voraz, para calcular la estimación
double calculo_voraz(vector<Objeto> const& objetos, double M,
                     Nodo const& X) {
   double hueco = M - X.peso, estimacion = X.beneficio;
   int i = X.k + 1;
   while (i < objetos.size() && objetos[i].peso <= hueco) {
      // podemos coger el objeto j entero
      hueco -= objetos[i].peso;
      estimacion += objetos[i].valor;
      ++i;
   }
   if (i < objetos.size())
      estimacion += (hueco / objetos[i].peso) * objetos[i].valor;
   return estimacion;
}
```

Implementación

```
// objetos ordenados de mayor a menor valor/peso
void mochila_rp(vector<Objeto> const& objetos, double M,
                vector<bool> & sol_mejor, double & beneficio_mejor) {
   int N = objetos.size();
   // se genera la raíz
   Nodo Y;
   Y.sol = vector<bool>(N);
   Y.k = -1;
   Y.peso = Y.beneficio = 0;
   Y.beneficio_est = calculo_voraz(objetos, M, Y);
   priority_queue<Nodo> cola;
   cola.push(Y);
   beneficio_mejor = -1;
```

Implementación

```
// búsqueda mientras pueda haber algo mejor
while (!cola.empty() && cola.top().beneficio_est > beneficio_mejor) {
   Y = cola.top(); cola.pop();
   Nodo X(Y);
   ++X.k;
   // probamos a meter el objeto en la mochila
   if (Y.peso + objetos[X.k].peso <= M) {
      X.sol[X.k] = true;
      X.peso = Y.peso + objetos[X.k].peso;
      X.beneficio = Y.beneficio + objetos[X.k].valor;
      X.beneficio_est = Y.beneficio_est;
      if (X.k == N-1) {
         sol_mejor = X.sol; beneficio_mejor = X.beneficio;
      } else cola.push(X);
}
```

Implementación

```
        // probar a no meter el objeto
        X.sol[X.k] = false;
        X.peso = Y.peso; X.beneficio = Y.beneficio;
        X.beneficio_est = calculo_voraz(objetos, M, X);
        if (X.beneficio_est > beneficio_mejor) {
           if (X.k == N-1) {
              sol_mejor = X.sol; beneficio_mejor = X.beneficio;
           } else
              cola.push(X);
        }
    }
}
```