.UBAfiuba
FACULTAD DE INGENIERÍA

Branch & Bound: Problema de la mochila

Teoría de Algoritmos I (75.29 / 95.06)
Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

Problema de la mochila

- **Contamos con:**
  - una mochila con una capacidad de K kilos
  - un subconjunto del conjunto E de “n” elementos
- **Cada elemento i tiene:**
  - un peso de k_i kilos
  - un valor de v_i.
- **Queremos seleccionar un subconjunto de E**
  - con el objetivo de maximizar la ganancia.
  - el peso total seleccionado no puede superar la capacidad de la mochila.

Valor por unidad del elemento

A cada elemento le calcularemos su valor por unidad u_i

Cociente entre el cociente entre su valor v_i y su peso k_i.

Un elemento con mayor valor por unidad que otro

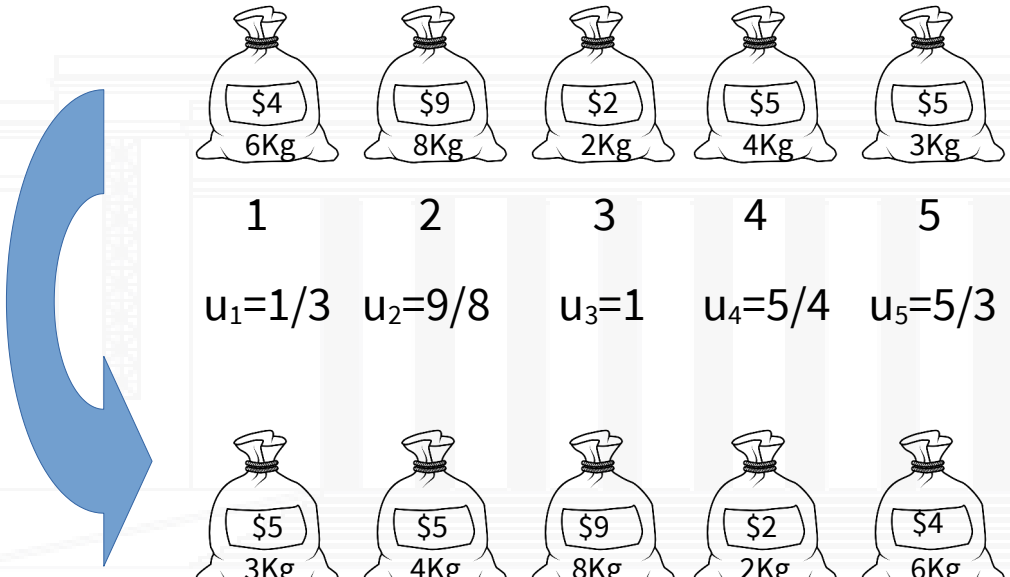
Produce una mayor ganancia ante misma cantidad de espacio ocupado

Ordenaremos de forma descendiente los elementos por valor unidad

asignaremos a cada elemento un identificador único (un valor entero) según este orden

Ejemplo: Valor por unidad

Supongamos la siguiente instancia



$u_1=1/3$ $u_2=9/8$ $u_3=1$ $u_4=5/4$ $u_5=5/3$

Árbol de estados del problema

Utilizaremos una estructura de árbol binario

cada nivel corresponde a determinar si el elemento i se incluye o no dentro de la mochila.

Utilizará el orden de elementos por valor por unidad

La raíz corresponde a la mochila vacía.

Cada nodo en el nivel “i tiene

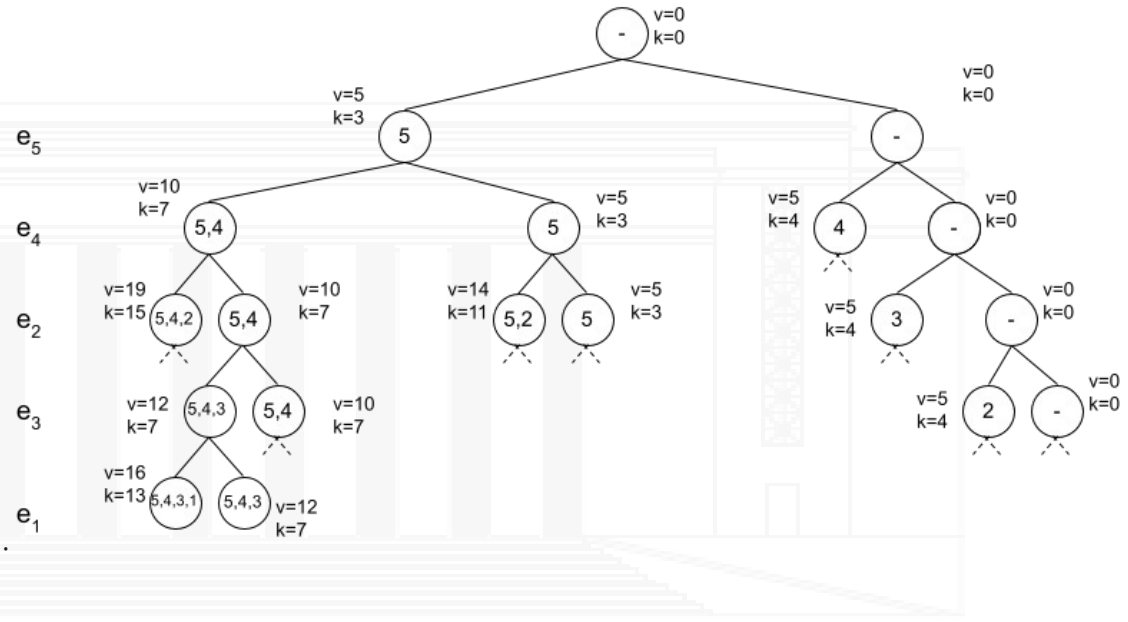
el peso cargado “k” en la mochila

el valor “v” de ganancia obtenido

dos descendientes (contiene o no el siguiente elemento).

Si tenemos n elementos en la mochila

la profundidad máxima corresponderá a n.



Función Costo “l”

Dado un nodo a una profundidad del árbol i

veremos cual es el elemento aun no evaluado de mayor valor por unidad (Dado el ordenamiento planteado, corresponde simplemente al elemento i+1)

Obtenemos cuál es el espacio disponible en la mochila (K-k).

Supondremos que

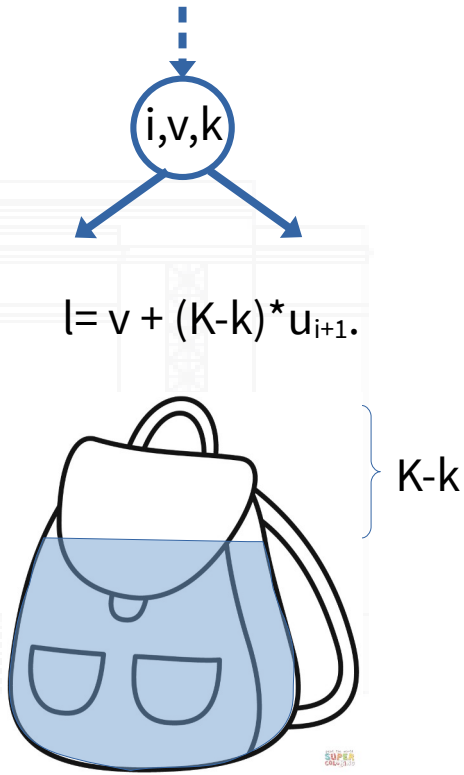
en el mejor de los casos todos ese espacio se llenará con el elemento i+1 (u otros posteriores con mismo valor por unidad).

Calcularemos la función costo de un nodo

Sumando el valor de la ganancia obtenida en el nodo actual “v” con la mayor ganancia hipotética posible

Corresponde al tope a la ganancia de cualquier estado del árbol que descienda del nodo.

Si este valor es inferior a la máxima ganancia encontrada, podemos realizar la poda.



Recorrido del árbol de estados

Para recorrer el árbol utilizaremos depth-first branch-and-bound.

- Comenzará por la raíz
- Inicialmente la mejor solución es la mochila vacia (k=0, v=0)

Dado el nodo actual

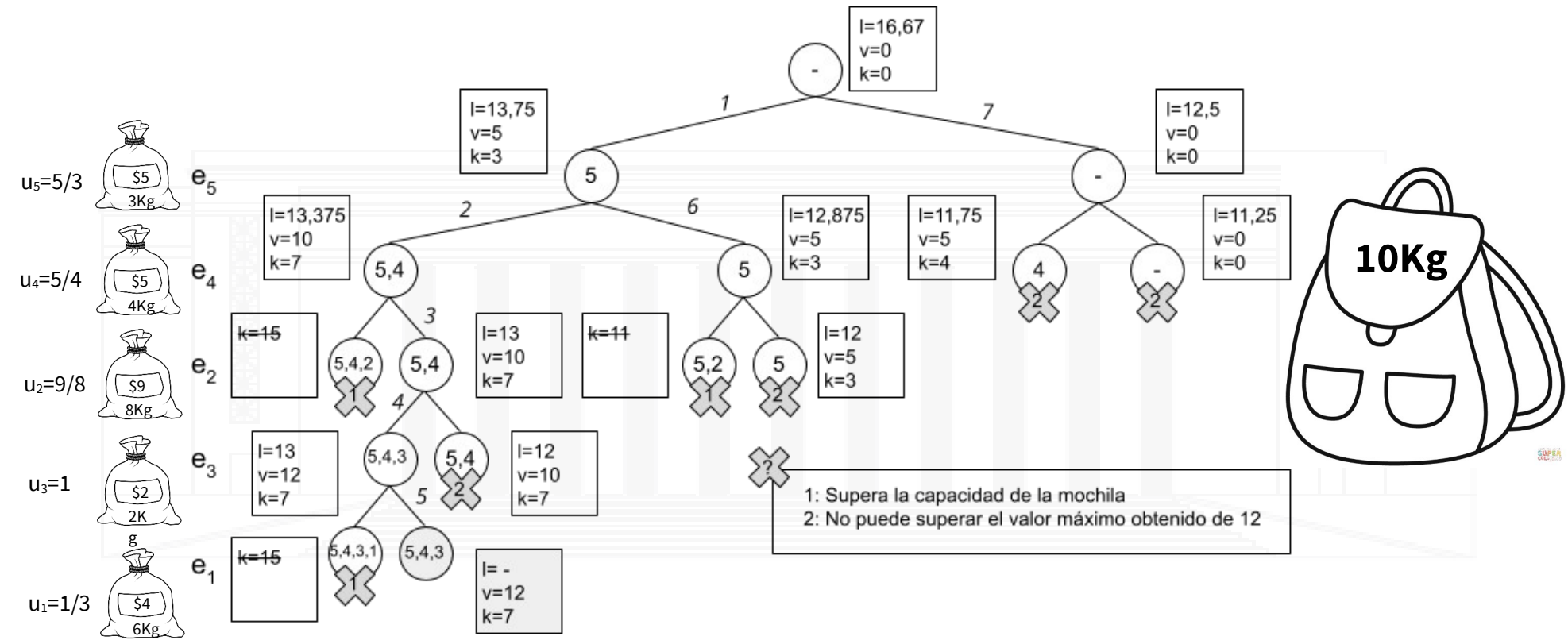
- Verificamos si es mejor que la mejor solución actual
- Expandimos cada uno de sus descendientes y calculamos su función costo

Dados sus descendientes no explorados que no superen la capacidad de la mochila y pueden tener una mejor solución a la actual

- Seleccionamos al mejor de ellos (mayor función costo) y lo definimos como el nodo actual

Si no quedan descendientes por explorar se regresa al nodo padre

Exploración del árbol en el ejemplo



Branch & Bound – Pseudocódigo

```
Sea mochila la mochila inicialmente vacia
Sea K la capacidad de la mochila
Sea elementos el listado de n elementos ordenados segun su valor por unidad
Sea nroelemento la posicion en el listado de elemento a evaluar
Sea mejorMochila la combinacion de la mochila que da mayor ganancia encontrada
Sea mejorGanancia la combinacion de la mochila que da mayor ganancia encontrada

nroelemento =1
mejorResultado = {}
mejorGanancia = 0
Backtrack(mochila,nroelemento)
```

Branch & Bound – Pseudocódigo

```
Backtrack (mochila, nro):
     Sea elemento el elemento en la posicion nro en elementos.

     Sea mochilaAmpliada = mochila ∪ elemento
     Sea descendientes los nodos descendientes de estadoActual
     Sea pesoActual el peso de los elementos en la mochila
     Sea gananciaActual la suma de los valores de los elementos en mochilaAmpliada
     Sea pesoAmpliado el peso de los elementos en la mochilaAmpiada
     Sea gananciaAmpliada la suma de los valores de los elementos en mochilaAmpliada
     Si nro==n
          Si pesoActual <= K y gananciaActual > mejorGanancia
               mejorMochila = mochila
               mejorGanancia = gananciaActual
          Si pesoAmpliado <= K y gananciaAmpliada > mejorGanancia
               mejorMochila = mochilaAmpliada
               mejorGanancia = gananciaAmpliada

     sino
```

Branch & Bound – Pseudocódigo

```
Sea elementoSig el elemento en la posicion nro+1 en elementos.
Sea valorUnSig el valor por unidad de elementoSig

Sea CotaActual = gananciaActual + (K -  pesoActual ) * valorUnSig
Sea CotaAmpliada = gananciaAmpliada + (K -  pesoAmpliado ) * valorUnSig
Sea opciones los estados descendientes posibles.

Si  pesoActual <= K y CotaActual > mejorGanancia
     Agregar mochila a descendientes con CotaActual

Si  pesoActual <= K y CotaActual > mejorGanancia
     Agregar mochilaAmpliada a descendientes con CotaAmpliada

Mientras existan mochilas en descendientes no explorados
     Sea mochilaDesc en descendientes aún no analizada con mayor cota
     Si cota > mejorGanancia
          Backtrack(mochilaDesc,nro+1)
```

Complejidad Temporal

En el peor de los casos se analiza 2^n estados del problema.

En cada estado se realizan procedimientos O(1) para los cálculos de los limites y costos

Actualizar la mejor solución encontrada es O(n)

En el peor de los casos se realiza ante cada agregado de un elemento

La complejdad temporal es la multiplicación de estos dos valores

Complejidad Espacial

Por la implementación recursiva

Por cada llamado en profundidad en el árbol incluimos el consumo de memoria adicional

La memoria utilizada

Es proporcional a la profundidad máxima de la recursión generada

La profundidad máxima es “n”.

En cada nivel de profundidad

Se agrega (o no) un elemento a la mochila

Se realizan cálculos que requieren O(1) de almacenamiento