Knaspsack Problem aproximado

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

Problema de la mochila

Sea

Conjunto de n items $X=\{x_1,...,x_n\}$

Cada item x_i contiene un valor v_i y un peso w_i

Mochila de tamaño W

Deseamos

Encontrar un subset $S \subseteq X$

tal que

la suma de los pesos de los elementos en S no supere W

Y la suma del valor de los elementos en S sea el máximo

Soluciones

Mediante programación dinámica

Hemos logrado un algoritmo pseudopolinomial O(nW)

Se ha demostrado

Que corresponde a un problema NP-Completo

Por lo que (A menos que P=NP)

No podemos encontrar un algoritmo de resolución totalmente polinomial

Si W y N es muy grande

El problema se torna intractable

Una solución aproximada...

Propondremos

Un algoritmo de aproximación

De tipo

Esquema de aproximación en tiempo polinómico

Utilizaremos un parámetro

ε que nos permitirá determinar la precisión deseada

Se ejecutara

En tiempo polinómico

Como parte de su ejecución

Utilizará programación dinámica

Una solución parametrizada y acotable

La solución de aproximación

Nos retornará un subconjunto de elementos S

Que no supere

Entre ellos el peso W

Con un valor total V

Que es igual o menor al valor máximo óptimo

Fijaremos el parámetro ε

para acotar la diferencia máxima entre el valor encontrado y el óptimo

Programación dinámica (recargada)

Necesitamos que el algoritmo

De programación dinámica utilice para hallar el optimo el Valor (y no el peso)

De esa forma podremos ajustar el parámetro V

Según nuestra conveniencia para aproximar el resultado

El algoritmo dividirá el problema

En subproblemas que se superponen para memorizar y evitar repetir cálculo

Subproblemas

Llamaremos

OPT(i,V)

Al subproblema de determinar

El menor peso que se puede obtener con los primeros i items cuyo valor iguale o supere el valor de al menos V en la mochila

Se calculará el subproblema para

I=0,…,n

$V=0,\dots,V_{max}$

Con $V_{max}=\sum_{j=1}^{n} v_j$ Valor equivalente a incluir todos los elementos en la mochila

Casos base

Para obtener un valor v=0

No hace falta poner ningún elemento

$OPT(i,v)=0$, $v \le 0$

Si tengo cero elementos

No puedo lograr ningún valor

(excepto si el valor es 0: Corresponde al caso anterior)

Para expresar la imposibilidad utilizaremos el ∞

(o un peso mayor la suma de los pesos de todos los elementos)

$OPT(v,i) = \infty$, $v>0$ $i=0$

Solapamiento de subproblemas

En un subproblema genérico OPT(i,v)

Pueden ocurrir 2 casos

Que el i-esimo problema no se encuentre en la solución

En ese caso buscamos el menor peso en lograr el valor v con los i-1 elementos anteriores → OPT(i-1,v)

Que el i-esimo problema se encuentre en la solución

En ese caso sumamos a la mochila W_i de pesos y el menor peso para valor $v-v_i$ con los i-1 elementos → OPT(i-,v-v_i)

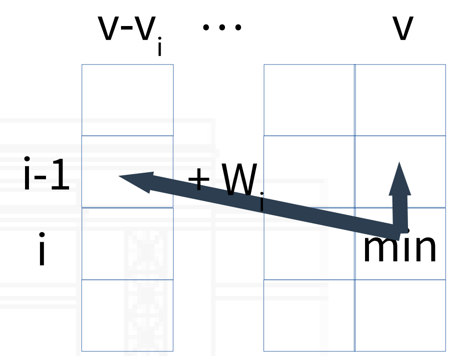
Como se desea minimizar el peso de la mochila

el optimo contendrá el menor de los 2 casos

Recurrencia

Podemos expresar la relación de recurrencia como

$$\begin{cases} 0 & , \text{si } v=0 \\ \infty & , \text{si i=0 } y\, v>0 \\ min\{OPT(i-1,v), w_i + OPT(i-1, max\{0, v-v_i\})\} & \end{cases}$$



Una vez que

tengo resueltos todos los subproblemas

El valor que maximiza el problema sera

El mayor u con u=0,..., v_{max}

que cumpla que OPT(n,u) ≤ W

Pseudocódigo

Complejidad

Temporal: $O(nV_{max})$

Espacial: $O(nV_{max})$

Para recobrar los elementos seleccionados

Debo almacenar para cada caso si se selecciono o no que el elemento esté en el optimo

Iterar desde el optimo para atrás reconstruyendo

```
Desde i=0 a n
    OPT[i][0] = 0

Desde v=1 a Vmax
    OPT[0][v] = +∞

Desde i=1 a n   // elementos
    Desde v=1 a Vmax   // valores

        enOptimo = w[i] + OPT[i-1,v-v[i]]
        noEnOptimo = OPT[i-1,v]

        si enOptimo < noEnOptimo
            OPT[l][p] = enOptimo
        sino
            OPT[l][p] = noEnOptimo

Desde v=Vmax a 0
    si OPT[n,v]<=W
        retornar OPT[n,v]
```

Acotando de forma mas conveniente

Si llamamos

$v^* = \max\{ V_i \}$ con $0<i\leq n$

Podemos acotar

$$V_{max} = \sum_{j=1}^{n} v_j \leq n v^*$$

Por lo tanto

La complejidad de la programación dinámica sera $O(n^2 v^*)$

(esta forma de expresarlo será ventajosa más adelante)

Lo que tenemos hasta ahora...

La solución es pseudo polinomial

depende de los valores v_i

Si v* es pequeño

Entonces $O(n^2v^*)$ “funcionará” en tiempo polinomial

Sino

Tendremos que aproximar.

Parámetro de redondeo

Ejemplo:

b=20	x_1	x_2	x_3
v_i	126	37	413
$\underline{v}_i$	140	40	420
$\overline{v}_i$	7	2	21

v*

Utilizaremos

El parámetro b de redondeo

Para cada item i

Calcularemos $\underline{v}_i = \lceil v_i / b \rceil * b$

Todos los valores de items resultantes

Son múltiplos de b → $v_i \leq \underline{v}_i \leq v_i + b$

Podemos resolver mediante programación dinámica utilizando

$\overline{v}_i = \lceil v_i / b \rceil$ ← nos asegura que sean valores enteros

No quedará v* mas pequeño

Resolución del parámetro

Resolveremos el problema

Utilizando los nuevos valores $\overline{v}_i$

El resultado obtenido

tiene el mismo set de elementos óptimos que utilizando vi

(mismo peso y un difieren en un factor de b)

Obtenemos los elementos de la solución aproximada

Su valor real será menor o igual al obtenido

Elección del parámetro b

Utilizaremos

ε para generar el parámetro b,

Con $0 < \varepsilon \leq 1$

Y por comodidad ε^{-1} un número entero

Un valor conveniente de b

$b = \varepsilon v^* / 2n$

(esta elección nos servirá para las próximas demostraciones)

Pseudocodigo

```
Obtener vmax

Definir b = ε vmax / 2n

Para cada elemento i
     Calcular v̄i con b

Resolver con programación dinámica con valores v̄i

Retornar el set de elementos encontrados
```

Complejidad temporal global

La programación dinámica se ejecuta en $O(n^2v^*)$

Con $v^* = \max\{ V_i \}$ con $0<i\leq n$

Si el item j es el de máximo valor

Entonces $v^* = \bar{v}_j = \lceil v_j / b \rceil$

Siendo que $b = \varepsilon v^* / 2n$

Entonces $v^* = 2n\varepsilon^{-1}$

Todo el proceso

será $O(n^3\varepsilon^{-1})$

Para un valor fijo de ε εl algoritmo se ejecuta en tiempo polinomial (!)

Margen de la aproximación

Llamaremos

S* cualquier una solución que satisface $\sum_{i \in S^*} w_i \leq W$

El algoritmo aproximado encuentra la solución optima S

Para los valores $\underline{v}_i$ aproximados (fueron redondeados para arriba)

$$OPT(S) = \sum_{i \in S} \underline{v}_i \geq \sum_{i \in S^*} \underline{v}_i$$

Se puede ver que

$$\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} \underline{v}_i \leq \sum_{i \in S} \underline{v}_i \leq \sum_{i \in S} b + v_i \leq nb + \sum_{i \in S} v_i$$

Por redondeo

Por ser optima por la aproximación

$v_i \leq \underline{v}_i \leq v_i + b$

Si Todos los elementos están en la solución

Si este fuese el máximo valor posible

$$\sum_{i \in S^*} v_i \leq nb + \sum_{i \in S} v_i$$

(!) La solución encontrada es como mucho nb menor al máximo valor posible

Expresando en función de ε

Como b = ε v* / 2n

$$nb+\sum_{i\in S} v_i=\frac{\varepsilon}{2}v^*+\sum_{i\in S} v_i$$

Entonces

$$\sum_{i\in S^*} v_i \le \frac{\varepsilon}{2}v^*+\sum_{i\in S} v_i$$

Como cualquier item entra en la mochila

Una posible solución S* = {x_j} con v_j=v*

$$v^*\le\frac{\varepsilon}{2}v^*+\sum_{i\in S} v_i \le \frac{v^*}{2}+\sum_{i\in S} v_i \Rightarrow \frac{v^*}{2}\le\sum_{i\in S} v_i \Rightarrow v^*\le 2\sum_{i\in S} v_i$$

Unificando

$$\sum_{i\in S^*} v_i \le \frac{\varepsilon}{2}v^*+\sum_{i\in S} v_i \le \frac{\varepsilon}{2}\left(2\sum_{i\in S} v_i\right)+\sum_{i\in S} v_i \Rightarrow \boxed{\sum_{i\in S^*} v_i\le(1+\varepsilon)\sum_{i\in S} v_i}$$

Conclusión

Si

S es la solución encontrara por el algoritmo de aproximación

S* es cualquier otra solución factible

Entonces

$$\sum_{i \in S^*} v_i \leq (1+\varepsilon) \sum_{i \in S} v_i$$

Por lo tanto,

Para cualquier $\varepsilon>0$, la solución aproximada encuentra una solución factible cuyo valor esta dentro de un factor $(1+\varepsilon)$ de la solución óptima

Y lo realiza en tiempo polinomial $O(n^3\varepsilon^{-1})$

.UBA fiuba

FACULTAD DE INGENIERÍA

Presentación realizada en Julio de 2020