Algoritmos de aproximação - Problema da Mochila

Marina Andretta

ICMC-USP

11 de novembro de 2015

Baseado nos livros Minicurso de Análise de Algoritmos, de P. Feofiloff; e Uma introdução sucinta a Algoritmos de Aproximação, de M. H. Carvalho, M. R. Cerioli, R. Dahab, P. Feofiloff, C. G. Fernandes, C. E. Ferreira, K. S. Guimarães, F. K. Miyazawa, J. C. Piña Jr., J. A. R. Soares e Y. Wakabayashi.

Problema da mochila

Trataremos agora de um problema de otimização bem conhecido: o problema da mochila (*knapsack problem*).

Problema $\textsc{Mochila}(m, n, v, w)$: Dados um número m em $\mathbb{Q}_{\geq}$, um número n em $\mathbb{Z}_{>}$, um número v_i em $\mathbb{Z}_{\geq}$ e um número w_i em $\mathbb{Q}_{\geq}$ para cada i em $\{1, ..., n\}$, encontrar um subconjunto S de $\{1, ..., n\}$ que maximize $v(S)$ sob a restrição $w(S) \leq m$.

Problema da mochila

Os números v_i e w_i podem ser interpretados como valor e peso, respectivamente, de um objeto i.

O número m pode ser interpretado como a capacidade de uma mochila, ou seja, o peso máximo que a mochila comporta.

O objetivo do problema é encontrar uma coleção de objetos mais valiosa possível que respeite a capacidade da mochila.

Algoritmo exato

Uma abordagem de programação dinâmica resolve o problema MOCHILA: basta construir uma tabela W, com W_{ij} o peso mínimo de um subconjunto de $\{1, ..., i\}$ cujo valor é pelo menos j. Ou seja,

$$W_{ij} := \min\{w(S) : S \subseteq \{1, ..., i\} \text{ e } v(S) \geq j\}.$$

Aqui, i varia de 0 a n e j de 0 ao valor ótimo $opt(m, n, v, w)$ do problema mais 1.

Se não há subconjunto de $\{1, ..., i\}$ de valor pelo menos j, dizemos que $W_{ij} = \infty$.

Algoritmo exato

Algoritmo MOCHILA-EXATO(m, n, v, w):

1 para i de 0 a n, faça $W_{i0} \leftarrow 0$;
2 faça $j \leftarrow 0$;
3 repita:
4 $j \leftarrow j + 1$;
5 $W_{0j} \leftarrow \infty$;
6 para i de 1 a n, faça
7 se $v_i \geq j$
8 então $W_{ij} \leftarrow \min\{W_{i-1,j}, w_i\}$;
9 senão $W_{ij} \leftarrow \min\{W_{i-1,j}, w_i + W_{i-1,j-v_i}\}$;
10 até que $W_{nj} > m$.
11 Seja S um subconjunto de $\{1, ..., n\}$
com $w(S) = W_{n,j-1}$ e $v(S) \geq j - 1$;
12 devolva S.

Algoritmo exato

Vejamos agora porque o algoritmo Mochila-Exata funciona.

A ideia das linhas 7 a 9 é a seguinte: suponha que S é um conjunto de peso mínimo dentre os que estão incluídos em $\{1, ..., i\}$ e têm valor pelo menos j.

Se $i \notin S$ então S é um conjunto de peso mínimo dentre os que estão incluídos em $\{1, ..., i-1\}$ e têm valor pelo menos j.

Se $i \in S$ então $S \setminus \{i\}$ é um conjunto de peso mínimo dentre os que estão incluídos em $\{1, ..., i-1\}$ e têm valor pelo menos $j - v_i$. (Se $v_i \geq j$ então $S = \{i\}$.)

Algoritmo exato

Para justificar a condição de parada na linha 10, observe que $W_{nj} \leq W_{nk}$ para todo $j \leq k$, já que todo subconjunto S de $\{1, ..., n\}$ que satisfaz $v(S) \geq k$ também satisfaz $v(S) \geq j$.

Assim, se $W_{nj} > m$, então $W_{nk} > m$ para todo $k > j$ e, portanto, podemos interromper os cálculos.

Algoritmo exato - exemplo

Considere a instância do problema MOCHILA com $m = 7$, $n = 5$, v e w dados na tabela a seguir.

i	1	2	3	4	5
v_i	2	1	3	4	1
w_i	4	2	1	2	2

Vejamos como a tabela W é calculada pelo algoritmo MOCHILA-EXATO.

Algoritmo exato - exemplo

W	0
0	0
1	0
2	0
3	0
4	0
5	0

Algoritmo exato - exemplo

W	0	1
0	0	∞
1	0	4
2	0	2
3	0	1
4	0	1
5	0	1

Algoritmo exato - exemplo

W	0	1	2
0	0	∞	∞
1	0	4	4
2	0	2	4
3	0	1	1
4	0	1	1
5	0	1	1

Algoritmo exato - exemplo

W	0	1	2	3
0	0	∞	∞	∞
1	0	4	4	∞
2	0	2	4	6
3	0	1	1	1
4	0	1	1	1
5	0	1	1	1

Algoritmo exato - exemplo

W	0	1	2	3	4
0	0	∞	∞	∞	∞
1	0	4	4	∞	∞
2	0	2	4	6	∞
3	0	1	1	1	3
4	0	1	1	1	2
5	0	1	1	1	2

Algoritmo exato - exemplo

W	0	1	2	3	4	5
0	0	∞	∞	∞	∞	∞
1	0	4	4	∞	∞	∞
2	0	2	4	6	∞	∞
3	0	1	1	1	3	5
4	0	1	1	1	2	3
5	0	1	1	1	2	3

Algoritmo exato - exemplo

W	0	1	2	3	4	5	6
0	0	∞	∞	∞	∞	∞	∞
1	0	4	4	∞	∞	∞	∞
2	0	2	4	6	∞	∞	∞
3	0	1	1	1	3	5	7
4	0	1	1	1	2	3	3
5	0	1	1	1	2	3	3

Algoritmo exato - exemplo

W	0	1	2	3	4	5	6	7
0	0	∞	∞	∞	∞	∞	∞	∞
1	0	4	4	∞	∞	∞	∞	∞
2	0	2	4	6	∞	∞	∞	∞
3	0	1	1	1	3	5	7	∞
4	0	1	1	1	2	3	3	3
5	0	1	1	1	2	3	3	3

Algoritmo exato - exemplo

W	0	1	2	3	4	5	6	7	8
0	0	∞	∞	∞	∞	∞	∞	∞	∞
1	0	4	4	∞	∞	∞	∞	∞	∞
2	0	2	4	6	∞	∞	∞	∞	∞
3	0	1	1	1	3	5	7	∞	∞
4	0	1	1	1	2	3	3	3	5
5	0	1	1	1	2	3	3	3	5

Algoritmo exato - exemplo

W	0	1	2	3	4	5	6	7	8	9
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	4	4	∞	∞	∞	∞	∞	∞	∞
2	0	2	4	6	∞	∞	∞	∞	∞	∞
3	0	1	1	1	3	5	7	∞	∞	∞
4	0	1	1	1	2	3	3	3	5	7
5	0	1	1	1	2	3	3	3	5	7

Algoritmo exato - exemplo

W	0	1	2	3	4	5	6	7	8	9	10
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	4	4	∞	∞	∞	∞	∞	∞	∞	∞
2	0	2	4	6	∞	∞	∞	∞	∞	∞	∞
3	0	1	1	1	3	5	7	∞	∞	∞	∞
4	0	1	1	1	2	3	3	3	5	7	9
5	0	1	1	1	2	3	3	3	5	7	9

Algoritmo exato - exemplo

W	0	1	2	3	4	5	6	7	8	9	10
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	4	4	∞	∞	∞	∞	∞	∞	∞	∞
2	0	2	4	6	∞	∞	∞	∞	∞	∞	∞
3	0	1	1	1	3	5	7	∞	∞	∞	∞
4	0	1	1	1	2	3	3	3	5	7	9
5	0	1	1	1	2	3	3	3	5	7	9

O valor ótimo dessa instância é 9 e existem duas soluções ótimas: $\{1, 3, 4\}$

Algoritmo exato - exemplo

W	0	1	2	3	4	5	6	7	8	9	10
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	0	4	4	∞	∞	∞	∞	∞	∞	∞	∞
2	0	2	4	6	∞	∞	∞	∞	∞	∞	∞
3	0	1	1	1	3	5	7	∞	∞	∞	∞
4	0	1	1	1	2	3	3	3	5	7	9
5	0	1	1	1	2	3	3	3	5	7	9

O valor ótimo dessa instância é 9 e existem duas soluções ótimas: $\{1, 3, 4\}$ e $\{2, 3, 4, 5\}$.

Algoritmo exato

O número de execuções das linhas 3 a 10 pode não ser polinomial em $\langle v \rangle$.

Por exemplo, se $n = 1$ e $m > v_1$, o número de execuções das linhas 3 a 10 é $v_1 + 1$ enquanto que $\langle v_1 \rangle = O(\log(v_1))$.

Algoritmo exato

O problema MOCHILA é NP-difícil.

Podemos dizer entretanto que o algoritmo MOCHILA-EXATO consome tempo proporcional ao número de componentes da matriz W.

Esse número é limitado por $(n+1)(\sigma_v+1)$, onde

$$\sigma_v := \sum_{i:w_i \leq m} v_i,$$

já que o valor ótimo do problema MOCHILA(m, n, v, w) não ultrapassa σ_v (com $\sigma_v = 0$ se todo $w_i > m$).

Algoritmo exato

As linhas 11 e 12 podem ser executadas em tempo $O(n + \sigma_v)$.

Assim o consumo total de tempo do algoritmo MOCHILA-EXATO é $O(n(\sigma_v + 1))$.

Um algoritmo de aproximação

Como o problema MOCHILA é NP-difícil, vamos ver dois algoritmos de aproximação distintos para resolvê-lo.

O algoritmo que descrevemos a seguir tem caráter "guloso" e dá preferência aos objetos de maior valor específico (v/w).

Um algoritmo de aproximação

Para simplificar a descrição do algoritmo, vamos supor que os objetos são dados em ordem decrescente de valor específico, ou seja, que

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}.$$

Sem perda de generalidade, vamos supor também que $0 < w_i \leq m$, para i de 1 a n.

Um algoritmo de aproximação

Algoritmo MOCHILA-QUASE-ÓTIMO(m, n, v, w):

```
1   faça S ← ∅;
2   faça s ← x ← 0;
3   faça k ← 1;
4   enquanto k ≤ n e s + w_k ≤ m faça
5       S ← S ∪ {k};
6       s ← s + w_k;
7       x ← x + v_k;
8       k ← k + 1;
9   se k > n ou x ≥ v_k
10      então devolva S;
11      senão devolva {k}.
```

Um algoritmo de aproximação - exemplo

Considere a mesma instância do problema MOCHILA usado para exemplificar a execução do algoritmo MOCHILA-EXATO. Ou seja, $m = 7$, $n = 5$, v e w dados na tabela a seguir.

k	1	2	3	4	5
i_k	3	4	1	2	5
v_{i_k}	3	4	2	1	1
w_{i_k}	1	2	4	2	2

Lembre-se que os itens devem estar ordenados por valor específico, então eles serão visitados pelo algoritmo MOCHILA-QUASE-ÓTIMO na ordem 3, 4, 1, 2, 5.

Um algoritmo de aproximação - exemplo

k	1	2	3	4
S	$\emptyset$	$\{3\}$	$\{3,4\}$	$\{3,4,1\}$
s	0	1	3	7
x	0	3	7	9

Como o próximo elemento a ser inserido em S seria o objeto $i_4 = 2$, que tem peso $w_2 = 2$, temos que $s + w_2 = 9 > 7 = m$ e, por isso, o algoritmo vai para a linha 9.

Como $x = 9 > 1 = v_2$, temos que o algoritmo devolve como solução $S = \{4, 3, 1\}$, com peso $s = 7$ e valor $x = 9$.

Um algoritmo de aproximação

Teorema 1: O algoritmo MOCHILA-QUASE-ÓTIMO é uma $\frac{1}{2}$-aproximação polinomial para o problema MOCHILA.

Demonstração: Considere uma instância MOCHILA(m, n, v, w). Vamos supor que $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq ... \geq \frac{v_n}{w_n}$ e $0 < w_i \leq m$, para i de 1 a n.

O bloco de linhas 4 a 8 determina o maior k tal que $w_1 + ... + w_{k-1} \leq m$.

No início da linha 9, $S = \{1, ..., k-1\}$, $s = w(S)$ e $x = v(S)$.

No início da linha 9, é claro que S é viável. Se $k > n$ então $S = \{1, ..., n\}$ e o algoritmo adota S como solução.

Neste caso, é evidente que $v(S) \geq \frac{1}{2} opt(m, n, v, w)$.

Um algoritmo de aproximação

Suponha agora que $k \leq n$ no início da linha 9.

Como $0 < w_i \leq m$, o conjunto $\{k\}$ é viável e o algoritmo adota como solução mais valiosa entre S e $\{k\}$.

Resta mostrar que $\max\{v(S), v_k\} \geq \frac{1}{2} opt(m, n, v, w)$.

Um algoritmo de aproximação

Primeiramente, note que

$$\max\{v(S), v_k\} \geq \frac{1}{2}(v(S) + v_k) = \frac{1}{2}v(R),$$

com $R := S \cup \{k\}$.

Vejamos agora um limitante para $v(R)$.

Um algoritmo de aproximação

Considere um conjunto viável qualquer S'. Temos que

$$\begin{aligned} v(R) - v(S') &= v(R \setminus S') - v(S' \setminus R) \\ &= \sum_{i \in R \setminus S'} v_i - \sum_{i \in S' \setminus R} v_i \\ &= \sum_{i \in R \setminus S'} \frac{v_i}{w_i} w_i - \sum_{i \in S' \setminus R} \frac{v_i}{w_i} w_i. \end{aligned}$$

Um algoritmo de aproximação

Como $v_i/w_i \geq v_k/w_k$ para todo i em R e $v_i/w_i \leq v_k/w_k$ para todo i no complemento de R, temos

$$\begin{aligned} v(R) - v(S') &\geq \frac{v_k}{w_k}\sum_{i\in R\setminus S'} w_i - \frac{v_k}{w_k}\sum_{i\in S'\setminus R} w_i. \\ &= \frac{v_k}{w_k} w(R \setminus S') - \frac{v_k}{w_k} w(S' \setminus R) \\ &= \frac{v_k}{w_k}(w(R) - w(S')). \end{aligned}$$

Um algoritmo de aproximação

Como $w(R) > m$ e $w(S') \leq m$,

$$\begin{aligned} v(R) - v(S') &\geq \frac{v_k}{w_k}(w(R) - w(S')). \\ &> \frac{v_k}{w_k}(m - m) = 0. \\ &\Rightarrow v(R) > v(S'). \end{aligned}$$

Já que S' é um conjunto viável arbitrário, $v(R) > opt(m, n, v, w)$.

Um algoritmo de aproximação

Portanto,

$$\max\{v(S), v_k\} \geq \frac{1}{2}v(R) > \frac{1}{2}opt(m, n, v, w).$$

Quanto ao consumo de tempo, é claro que o algoritmo é polinomial. Mais especificamente, ele consome tempo $\Theta(n \log(n))$.

Outro algoritmo de aproximação

Seja ϵ um número racional no intervalo aberto $(0, 1)$.

Vamos ver agora como usar o MOCHILA-EXATO para obter uma $(1 - \epsilon)$-aproximação polinomial para o problema MOCHILA.

O seguinte algoritmo, proposto por Ibarra e Kim, faz uma mudança de escala nos valores de uma instância (m, n, v, w) do problema, obtendo uma outra instância para a qual o algoritmo MOCHILA-EXATO consome tempo polinomial em m, n, $\langle v \rangle$, $\langle w \rangle$.

Algoritmo

Algoritmo $\textsc{Mochila-IK}_\epsilon(m, n, v, w)$:

1 se $w_i > m$ para todo i

2 então devolva $\emptyset$;

3 senão $\mathcal{V} \leftarrow \max_{i:w_i \leq m} v_i$;

4 faça $\lambda \leftarrow \epsilon\mathcal{V}/n$;

5 para i de 1 a n, faça $u_i \leftarrow \lfloor v_i/\lambda \rfloor$;

6 $S \leftarrow \textsc{Mochila-Exato}(m, n, u, w)$;

7 devolva S.

Outro algoritmo de aproximação

Na linha 5 do algoritmo, $\lfloor x \rfloor$ é o maior inteiro que não excede x.

Como S é uma solução ótima do problema $\textsc{Mochila}(m, n, u, w)$, temos que $\sum_{i \in S} w_i \leq m$, ou seja, S é uma solução viável do problema $\textsc{Mochila}(m, n, v, w)$.

O valor do objeto mais valioso cujo peso não excede a capacidade da mochila, que é o número $\mathcal{V}$, é um limitante inferior para o valor ótimo do problema. Ou seja,

$$opt(m, n, v, w) \geq \mathcal{V}.$$

Outro algoritmo de aproximação

Teorema 2: O algoritmo MOCHILA-IK$_\epsilon$ é uma $(1-\epsilon)$-aproximação polinomial para o problema MOCHILA.

Demonstração: O conjunto S na linha 6 do algoritmo é uma solução ótima do problema MOCHILA(m, n, u, w). Seja S^* uma solução ótima do MOCHILA(m, n, v, w).

Na linha 5 do algoritmo, definimos u_i como $\lfloor v_i/\lambda \rfloor$. Ou seja, $u_i \leq v_i/\lambda$.

Outro algoritmo de aproximação

Então

$$\begin{aligned}
\sum_{i\in S} v_i &\geq \lambda \sum_{i\in S} u_i \\
&\geq \lambda \sum_{i\in S^*} u_i \\
&\geq \lambda \sum_{i\in S^*} \left(\frac{v_i}{\lambda} - 1\right) \\
&= \lambda \sum_{i\in S^*} \frac{v_i}{\lambda} - \lambda \sum_{i\in S^*} 1 \\
&= v(S^*) - \lambda |S^*| \\
&\geq opt(m, n, v, w) - \lambda n \\
&= opt(m, n, v, w) - \epsilon \mathcal{V}.
\end{aligned}$$

Outro algoritmo de aproximação

Como $\mathcal{V} \leq opt(m, n, v, w)$,

$$\sum_{i \in S} v_i \geq (1 - \epsilon) opt(m, n, v, w).$$

Quanto ao tempo de execução do algoritmo, a linha 6 consome tempo $O(n(\sigma_u + 1))$, com $\sigma_u := \sum_{i:w_i \leq m} u_i$.

Como $u_i \leq v_i/\lambda \leq n/\epsilon$, para todo i tal que $w_i \leq m$, temos $\sigma_u \leq n^2/\epsilon$.

Portanto, o $\textsc{Mochila-IK}_\epsilon$ consome tempo $O(n^3/\epsilon)$.

Outro algoritmo de aproximação

Note que o algoritmo $\textsc{Mochila-IK}_\epsilon$ nos fornece, para cada ϵ racional no intervalo $(0, 1)$, uma $(1 - \epsilon)$-aproximação que consome tempo polinomial em n/ϵ para resolver a instância $\textsc{Mochila}(m, n, v, w)$.

Assim, $\textsc{Mochila-IK}_\epsilon$ é conhecido um esquema de aproximação plenamente polinomial (*fully polynomial-time approximation scheme*).