O algoritmo MINCC-CHVÁTAL pode produzir coberturas de custo arbitrariamente próximo de $H_n \operatorname{opt}(E, \mathcal{S}, c)$, onde $n := |E|$. Mais precisamente, para cada ε positivo, existe uma instância do problema para a qual o algoritmo produz uma cobertura de custo $H_n \operatorname{opt}(E, \mathcal{S}, c)/(1+\varepsilon)$: basta tomar $E := \{1, \ldots, n\}$, $\mathcal{S} := \{E, \{1\}, \ldots, \{n\}\}$, $c_E := 1+\varepsilon$ e $c_{\{i\}} := 1/i$ para cada i. Uma cobertura de custo mínimo é $\{E\}$ e o custo desta cobertura é $1+\varepsilon$. Por outro lado, o algoritmo MINCC-CHVÁTAL produz a cobertura $\{\{1\}, \ldots, \{n\}\}$, cujo custo é H_n.

Assintoticamente, o algoritmo MINCC-CHVÁTAL tem a melhor razão possível para o problema MINCC, pois sabe-se [RS97] que $H_n \leq 1 + \ln n$ e existe uma constante positiva α para a qual não existe algoritmo com razão de aproximação menor que $\alpha \ln n$, com $n = |E|$, para o MINCC $(E, \mathcal{S}, c)$, a menos que P = NP. Veremos mais sobre isso no capítulo 8.

2.3 Mochila

Nesta seção tratamos de um outro problema de otimização bem conhecido: o **problema da mochila** (*knapsack problem*). Esse problema tem importantes aplicações, como por exemplo o carregamento ótimo de *containers*. Podemos enunciá-lo da seguinte maneira.

> **Problema** 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 $\{1, \ldots, n\}$, encontrar um subconjunto S de $\{1, \ldots, n\}$ que maximize $v(S)$ sob a restrição $w(S) \leq m$.*

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 então encontrar uma coleção de objetos a mais valiosa possível que respeite a capacidade da mochila.[1]

Uma abordagem de programação dinâmica [CLR92] resolve o problema MOCHILA: basta construir uma tabela W onde W_{ij} é o peso mínimo de um subconjunto de $\{1, \ldots, i\}$ cujo valor é pelo menos j:

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

[1] Convém não confundir o problema MOCHILA com a sua variante fracionária [CLR92], que permite escolher qualquer fração de um objeto.

Aqui i varia de 0 a n e j de 0 ao valor ótimo $\mathrm{opt}(m, n, v, w)$ do problema mais 1. Se não há subconjunto de $\{1, \ldots, i\}$ de valor pelo menos j, então $W_{ij} = \infty$. Abaixo, segue o algoritmo e, na figura 2.2, um exemplo.

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

```
 1  para i de 0 a n faça W_{i0} ← 0
 2  j ← 0
 3  repita
 4      j ← j + 1
 5      W_{0j} ← ∞
 6      para i de 1 a n faça
 7          se v_i ≥ j
 8              então W_{ij} ← min {W_{i−1,j} , w_i}
 9              senão W_{ij} ← 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}
12      com w(S) = W_{n,j−1} e v(S) ≥ j − 1
13  devolva S
```

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

Figura 2.2: Aplicação do algoritmo MOCHILA-EXATO à instância $m = 7$, $n = 5$, $v = (2, 1, 3, 4, 1)$, $w = (4, 2, 1, 2, 2)$ do problema MOCHILA. A figura exibe a tabela W que o algoritmo calcula. O valor ótimo dessa instância é 9 e existem duas soluções ótimas: $\{1, 3, 4\}$ e $\{2, 3, 4, 5\}$.

As linhas 7 a 9 são fáceis de entender. Suponha que S é um conjunto de peso mínimo dentre os que estão incluídos em $\{1, \ldots, i\}$ e têm valor pelo menos j. Se $i \notin S$ então S é um conjunto de peso mínimo dentre os que estão incluídos em $\{1, \ldots, i-1\}$ 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 $\{1, \ldots, i-1\}$ e têm valor pelo menos $j - v_i$. (Se $v_i \geq j$ então $S = \{i\}$.)

Para justificar a condição de parada na linha 10, observe que $W_{nj} \leq W_{nk}$ para todo $k \geq j$, uma vez que todo subconjunto S de $\{1, \ldots, n\}$ 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.

Infelizmente, o número de execuções das linhas 3 a 10 pode não ser polinomial em $\langle v \rangle$. Por exemplo, se $n = 1$ e $m > v_1$, esse número é $v_1 + 1$ enquanto que $\langle v_1 \rangle = O(\log v_1)$. A bem da verdade, o problema MOCHILA é NP-difícil [GJ79]. 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

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

já que o valor ótimo do problema MOCHILA (m, n, v, w) não ultrapassa σ_v (com $\sigma_v = 0$ se todo $w_i > m$). Não é difícil verificar que as linhas 11 a 12 podem ser executadas em tempo $O(n + \sigma_v)$. Assim o consumo total de tempo do algoritmo MOCHILA-EXATO é $O(n(\sigma_v + 1))$.

Esquema de aproximação

Seja ε um número racional no intervalo aberto $(0, 1)$. Vamos ver agora como usar o MOCHILA-EXATO para obter uma $(1-\varepsilon)$-aproximação polinomial para o problema MOCHILA. O seguinte algoritmo, proposto por Ibarra e Kim [IK75], 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 $m, n, \langle v \rangle$ e $\langle w \rangle$.

Algoritmo MOCHILA-IK$_\varepsilon$ (m, n, v, w)

```
1   se w_i > m para todo i
2       então devolva ∅
3       senão ϑ ← max_{i : w_i ≤ m} v_i
4             λ ← εϑ/n
5             para i de 1 a n faça u_i ← ⌊v_i/λ⌋
6             S ← MOCHILA-EXATO (m, n, u, w)
7             devolva S
```

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

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

O valor do objeto mais valioso cujo peso não excede a capacidade da mochila, que é o número ϑ, é uma delimitação inferior para o valor ótimo do problema:

$$\text{opt}(m, n, v, w) \geq \vartheta \,. \tag{2.3}$$

Essa delimitação é usada na análise do algoritmo.

Teorema 2.3: *O algoritmo* MOCHILA-IK$_\varepsilon$ *é uma* $(1-\varepsilon)$*-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). Se S^* é uma solução ótima do MOCHILA(m, n, v, w), então

$$\begin{aligned}
\textstyle\sum_{i \in S} v_i &\geq \lambda \textstyle\sum_{i \in S} u_i \\
&\geq \lambda \textstyle\sum_{i \in S^*} u_i && (2.4)\\
&\geq \lambda \textstyle\sum_{i \in S^*} (\frac{v_i}{\lambda} - 1) && (2.5)\\
&= v(S^*) - \lambda\, |S^*| \\
&\geq v(S^*) - \lambda\, n \\
&= v(S^*) - \varepsilon \vartheta \\
&\geq (1 - \varepsilon)\, \text{opt}(m, n, v, w) \,. && (2.6)
\end{aligned}$$

A desigualdade (2.4) vale pois S é uma solução ótima do problema MOCHILA(m, n, u, w). Já (2.5) vale pois $u_i > v_i/\lambda - 1$ para todo i em S^*. Finalmente, (2.6) vale por (2.3).

Quanto ao tempo de execução do algoritmo, temos o seguinte. A linha 6 consome tempo $O(n(\sigma_u + 1))$, onde $\sigma_u := \sum_{i\,:\,w_i \leq m} u_i$. Mas $u_i \leq v_i/\lambda \leq n/\varepsilon$, para todo i tal que $w_i \leq m$. Assim $\sigma_u \leq n^2/\varepsilon$ e portanto o MOCHILA-IK$_\varepsilon$ consome tempo $O(n^3/\varepsilon)$. □

Note que o algoritmo MOCHILA-IK$_\varepsilon$ é um esquema que nos fornece, para cada ε racional no intervalo $(0, 1)$, uma $(1-\varepsilon)$-aproximação que consome tempo polinomial em n/ε para resolver a instância MOCHILA (m, n, v, w). Assim, MOCHILA-IK$_\varepsilon$ é conhecido um esquema de aproximação plenamente polinomial (*fully polynomial-time approximation scheme*) (capítulo 8).