21. Dynamic Programming III

FPTAS [Ottman/Widmayer, Kap. 7.2, 7.3, Cormen et al, Kap. 15,35.5]

Approximation

Let $\varepsilon \in (0,1)$ given. Let I_{opt} an optimal selection.

No try to find a valid selection I with

$$\sum_{i \in I} v_i \geq (1-\varepsilon) \sum_{i \in I_{\text{opt}}} v_i.$$

Sum of weights may not violate the weight limit.

Different formulation of the algorithm

Before: weight limit $w \to$ maximal value v

Reversed: value $v \to$ minimal weight w

$\Rightarrow$ **alternative table** $g[i,v]$ provides the minimum weight with

- a selection of the first i items ($0 \leq i \leq n$) that
- provide a value of exactly v ($0 \leq v \leq \sum_{i=1}^{n} v_i$).

Computation

Initially

- $g[0,0] \leftarrow 0$
- $g[0,v] \leftarrow \infty$ (Value v cannot be achieved with 0 items.).

Computation

$$g[i,v] \leftarrow \begin{cases} g[i-1,v] & \text{falls } v < v_i \\ \min\{g[i-1,v], g[i-1,v-v_i]+w_i\} & \text{sonst.} \end{cases}$$

incrementally in i and for fixed i increasing in v.

Solution can be found at largest index v with $g[n,v] \leq w$.

Example

$E = \{(2,3),(4,5),(1,1)\}$

$v \longrightarrow$

$i \downarrow$	0	1	2	3	4	5	6	7	8	9
$\emptyset$	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
$(2,3)$	0	∞	∞	2	∞	∞	∞	∞	∞	∞
$(4,5)$	0	∞	∞	2	∞	4	∞	∞	6	∞
$(1,1)$	0	1	∞	2	3	4	5	∞	6	7

Read out the solution: if $g[i,v] = g[i-1,v]$ then item i unused and continue with $g[i-1,v]$ otherwise used and continue with $g[i-1,b-v_i]$.

The approximation trick

Pseduopolynomial run time gets polynmial if the number of occuring values can be bounded by a polynom of the input length.

Let $K > 0$ be chosen *appropriately*. Replace values v_i by "rounded values" $\tilde{v}_i = \lfloor v_i/K \rfloor$ delivering a new input $E' = (w_i, \tilde{v}_i)_{i=1\ldots n}$.

Apply the algorithm on the input E' with the same weight limit W.

Idea

Example $K = 5$

Values

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \ldots, 98, 99, 100$$
$$\rightarrow$$
$$0, 0, 0, 0, 1, 1, 1, 1, 1, 2, \ldots, 19, 19, 20$$

Obviously less different values

Properties of the new algorithm

- Selection of items in E' is also admissible in E. Weight remains unchanged!
- Run time of the algorithm is bounded by $\mathcal{O}(n^2 \cdot v_{\max}/K)$ $(v_{\max} := \max\{v_i | 1 \le i \le n\})$

How good is the approximation?

It holds that

$$v_i - K \leq K \cdot \left\lfloor \frac{v_i}{K} \right\rfloor = K \cdot \tilde{v}_i \leq v_i$$

Let I'_{opt} be an optimal solution of E'. Then

$$\left(\sum_{i \in I_{\text{opt}}} v_i\right) - n \cdot K \overset{|I_{\text{opt}}| \leq n}{\leq} \sum_{i \in I_{\text{opt}}} (v_i - K) \leq \sum_{i \in I_{\text{opt}}} (K \cdot \tilde{v}_i) = K \sum_{i \in I_{\text{opt}}} \tilde{v}_i$$
$$\underset{I'_{\text{opt}}\text{optimal}}{\leq} K \sum_{i \in I'_{\text{opt}}} \tilde{v}_i = \sum_{i \in I'_{\text{opt}}} K \cdot \tilde{v}_i \leq \sum_{i \in I'_{\text{opt}}} v_i.$$

Choice of K

Requirement:

$$\sum_{i \in I'} v_i \geq (1 - \varepsilon) \sum_{i \in I_{\text{opt}}} v_i.$$

Inequality from above:

$$\sum_{i \in I'_{\text{opt}}} v_i \geq \left(\sum_{i \in I_{\text{opt}}} v_i\right) - n \cdot K$$

thus: $K = \varepsilon \frac{\sum_{i \in I_{\text{opt}}} v_i}{n}$.

Choice of K

Choose $K = \varepsilon \frac{\sum_{i \in I_{\text{opt}}} v_i}{n}$. The optimal sum is unknown. Therefore we choose $K' = \varepsilon \frac{v_{\max}}{n}$.[34]

It holds that $v_{\max} \leq \sum_{i \in I_{\text{opt}}} v_i$ and thus $K' \leq K$ and the approximation is even slightly better.

The run time of the algorithm is bounded by

$$\mathcal{O}(n^2 \cdot v_{\max}/K') = \mathcal{O}(n^2 \cdot v_{\max}/(\varepsilon \cdot v_{\max}/n)) = \mathcal{O}(n^3/\varepsilon).$$

[34] We can assume that items i with $w_i > W$ have been removed in the first place.

FPTAS

Such a family of algorithms is called an *approximation scheme*: the choice of ε controls both running time and approximation quality.

The runtime $\mathcal{O}(n^3/\varepsilon)$ is a polynom in n and in $\frac{1}{\varepsilon}$. The scheme is therefore also called a *FPTAS - Fully Polynomial Time Approximation Scheme*

22. Greedy Algorithms

Fractional Knapsack Problem, Huffman Coding [Cormen et al, Kap. 16.1, 16.3]

The Fractional Knapsack Problem

set of $n \in \mathbb{N}$ items $\{1, \dots, n\}$ Each item i has value $v_i \in \mathbb{N}$ and weight $w_i \in \mathbb{N}$. The maximum weight is given as $W \in \mathbb{N}$. Input is denoted as $E = (v_i, w_i)_{i=1,\dots,n}$.

Wanted: Fractions $0 \le q_i \le 1$ ($1 \le i \le n$) that maximise the sum $\sum_{i=1}^n q_i \cdot v_i$ under $\sum_{i=1}^n q_i \cdot w_i \le W$.

Greedy heuristics

Sort the items decreasingly by value per weight v_i/w_i.

Assumption $v_i/w_i \ge v_{i+1}/w_{i+1}$

Let $j = \max\{0 \le k \le n : \sum_{i=1}^k w_i \le W\}$. Set

- $q_i = 1$ for all $1 \le i \le j$.
- $q_{j+1} = \frac{W - \sum_{i=1}^j w_i}{w_{j+1}}$.
- $q_i = 0$ for all $i > j + 1$.

That is fast: $\Theta(n \log n)$ for sorting and $\Theta(n)$ for the computation of the q_i.

Correctness

Assumption: optimal solution (r_i) ($1 \le i \le n$).

The knapsack is full: $\sum_i r_i \cdot w_i = \sum_i q_i \cdot w_i = W$.

Consider k: smallest i with $r_i \ne q_i$ Definition of greedy: $q_k > r_k$. Let $x = q_k - r_k > 0$.

Construct a new solution (r_i'): $r_i' = r_i \forall i < k$. $r_k' = q_k$. Remove weight $\sum_{i=k+1}^n \delta_i = x \cdot w_k$ from items $k+1$ to n. This works because $\sum_{i=k}^n r_i \cdot w_i = \sum_{i=k}^n q_i \cdot w_i$.

Correctness

$$\begin{aligned}\sum_{i=k}^{n} r'_i v_i &= r_k v_k + x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^{n} (r_i w_i - \delta_i)\frac{v_i}{w_i}\\ &\geq r_k v_k + x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^{n} r_i w_i \frac{v_i}{w_i} - \delta_i \frac{v_k}{w_k}\\ &= r_k v_k + x w_k \frac{v_k}{w_k} - x w_k \frac{v_k}{w_k} + \sum_{i=k+1}^{n} r_i w_i \frac{v_i}{w_i} = \sum_{i=k}^{n} r_i v_i.\end{aligned}$$

Thus (r'_i) is also optimal. Iterative application of this idea generates the solution (q_i).

Huffman-Codes

Goal: memory-efficient saving of a sequence of characters using a binary code with code words..

Example

File consisting of 100.000 characters from the alphabet $\{a, \ldots, f\}$.

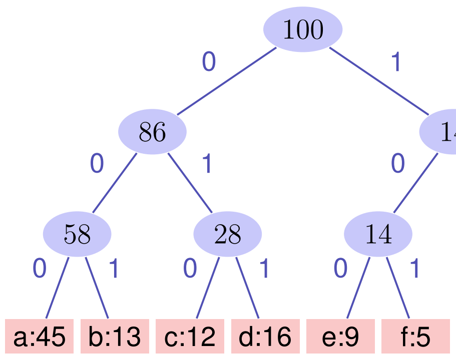
	a	b	c	d	e	f
Frequency (Thousands)	45	13	12	16	9	5
Code word with fix length	000	001	010	011	100	101
Code word variable length	0	101	100	111	1101	1100

File size (code with fix length): 300.000 bits.
File size (code with variable length): 224.000 bits.

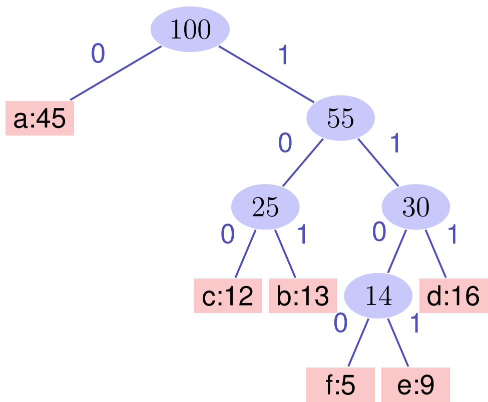
Huffman-Codes

- Consider prefix-codes: no code word can start with a different codeword.
- Prefix codes can, compared with other codes, achieve the optimal *data compression* (without proof here).
- Encoding: concatenation of the code words without stop character (difference to morsing).
  $affe \rightarrow 0 \cdot 1100 \cdot 1100 \cdot 1101 \rightarrow 0110011001101$
- Decoding simple because prefixcode
  $0110011001101 \rightarrow 0 \cdot 1100 \cdot 1100 \cdot 1101 \rightarrow affe$

Code trees



Code words with fixed length



Code words with variable length

Properties of the Code Trees

- An optimal coding of a file is alway represented by a complete binary tree: every inner node has two children.
- Let C be the set of all code words, $f(c)$ the frequency of a codeword c and $d_T(c)$ the depth of a code word in tree T. Define the cost of a tree as

$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c).$$

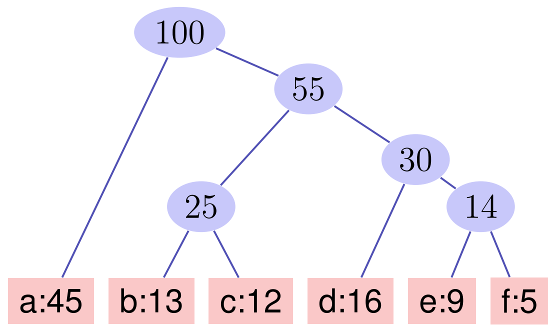
  (cost = number bits of the encoded file)

In the following a code tree is called optimal when it minimizes the costs.

Algorithm Idea

Tree construction bottom up

- Start with the set C of code words
- Replace iteriatively the two nodes with smallest frequency by a new parent node.



Algorithm Huffman(C)

Input : code words $c \in C$
Output : Root of an optimal code tree

$n \leftarrow |C|$
$Q \leftarrow C$
for $i = 1$ **to** $n - 1$ **do**
  allocate a new node z
  z.left $\leftarrow$ ExtractMin(Q)   // extract word with minimal frequency.
  z.right $\leftarrow$ ExtractMin(Q)
  z.freq $\leftarrow$ z.left.freq $+$ z.right.freq
  Insert(Q, z)
return ExtractMin(Q)

Analyse

Use a heap: build Heap in $\mathcal{O}(n)$. Extract-Min in $O(\log n)$ for n Elements. Yields a runtime of $O(n \log n)$.

The greedy approach is correct

Theorem

Let x, y be two symbols with smallest frequencies in C and let $T'(C')$ be an optimal code tree to the alphabet $C' = C - \{x, y\} + \{z\}$ with a new symbol z with $f(z) = f(x) + f(y)$. Then the tree $T(C)$ that is constructed from $T'(C')$ by replacing the node z by an inner node with children x and y is an optimal code tree for the alphabet C.

Proof

It holds that $f(x) \cdot d_T(x) + f(y) \cdot d_T(y) = (f(x) + f(y)) \cdot (d_{T'}(z) + 1) = f(z) \cdot d_{T'}(x) + f(x) + f(y)$. Thus $B(T') = B(T) - f(x) - f(y)$.

Assumption: T is not optimal. Then there is an optimal tree T'' with $B(T'') < B(T)$. We assume that x and y are brothers in T''. Let T''' be the tree where the inner node with children x and y is replaced by z. Then it holds that
$B(T''') = B(T'') - f(x) - f(y) < B(T) - f(x) - f(y) = B(T')$.
Contradiction to the optimality of T'.

The assumption that x and y are brothers in T'' can be justified because a swap of elements with smallest frequency to the lowest level of the tree can at most decrease the value of B.