

22. Dynamic Programming III

FPTAS [Ottman/Widmayer, Kap. 7.2, 7.3, Cormen et al, Kap. 15,35.5], Optimal Search Tree [Ottman/Widmayer, Kap. 5.7]

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 $w \rightarrow$ maximal value v

Reversed: value $v \rightarrow$ 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

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

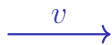
incrementally in i and for fixed i increasing in v .

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

Example

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

0 1 2 3 4 5 6 7 8 9



Example

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

		\xrightarrow{v}									
		0	1	2	3	4	5	6	7	8	9
\emptyset		0	∞	∞	∞	∞	∞	∞	∞	∞	∞

i
 \downarrow

Example

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

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

$v \rightarrow$

$i \downarrow$

Example

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

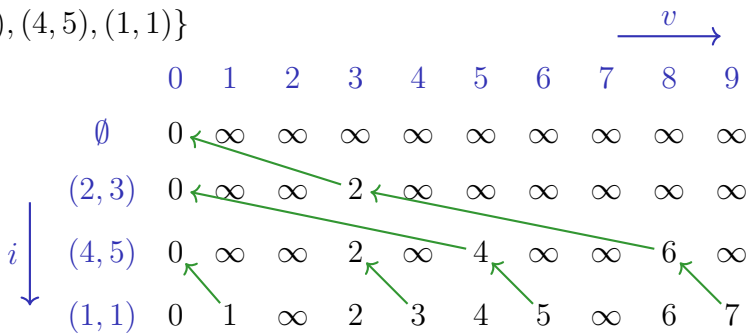
		0	1	2	3	4	5	6	7	8	9
	\emptyset	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
	(2, 3)	0	∞	∞	2	∞	∞	∞	∞	∞	∞
i	(4, 5)	0	∞	∞	2	∞	4	∞	∞	6	∞

$v \rightarrow$

$i \downarrow$

Example

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



Example

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

		$v \rightarrow$									
		0	1	2	3	4	5	6	7	8	9
$i \downarrow$	\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

Pseudopolynomial run time gets polynomial if the number of occurring values can be bounded by a polynomial 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 $E' = (w_i, \tilde{v}_i)_{i=1\dots n}$.

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, ..., 98, 99, 100

→

0, 0, 0, 0, 1, 1, 1, 1, 1, 2, ..., 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 $\mathcal{O}(n^2 \cdot v_{\max}/K)$
($v_{\max} := \max\{v_i | 1 \leq i \leq n\}$)

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

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

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 $K' = \varepsilon \frac{v_{\max}}{n}$.³⁸

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

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

³⁸We can assume that items i with $w_i > W$ have been removed in the first place.

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 polynomial in n and in $\frac{1}{\varepsilon}$. The scheme is therefore also called a **FPTAS - Fully Polynomial Time Approximation Scheme**

22.2 Optimale Suchbäume

Optimal binary Search Trees

Given: search probabilities p_i for each key k_i ($i = 1, \dots, n$) and q_i of each interval d_i ($i = 0, \dots, n$) between search keys of a binary search tree.

$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1.$$

Optimal binary Search Trees

Given: search probabilities p_i for each key k_i ($i = 1, \dots, n$) and q_i of each interval d_i ($i = 0, \dots, n$) between search keys of a binary search tree.

$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1.$$

Wanted: optimal search tree T with key depths $\text{depth}(\cdot)$, that minimizes the expected search costs

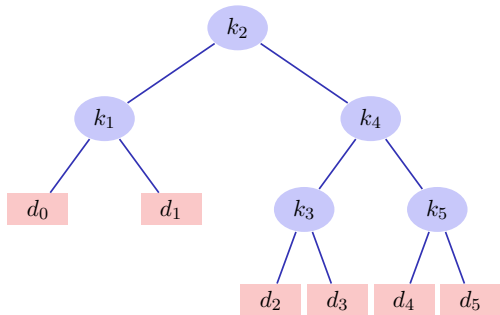
$$\begin{aligned} C(T) &= \sum_{i=1}^n p_i \cdot (\text{depth}(k_i) + 1) + \sum_{i=0}^n q_i \cdot (\text{depth}(d_i) + 1) \\ &= 1 + \sum_{i=1}^n p_i \cdot \text{depth}(k_i) + \sum_{i=0}^n q_i \cdot \text{depth}(d_i) \end{aligned}$$

Example

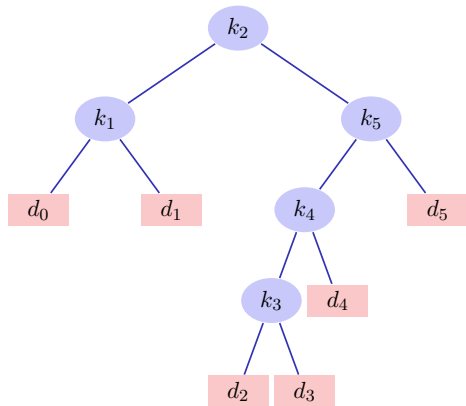
Expected Frequencies

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Example



Search tree with expected costs
2.8



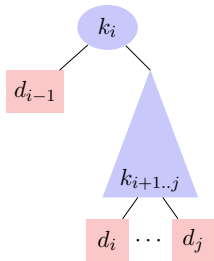
Search tree with expected costs
2.75

Structure of a optimal binary search tree

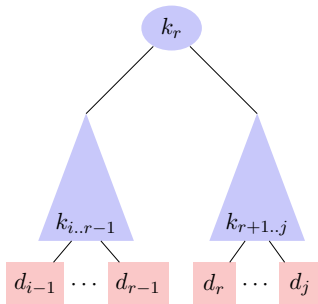
- Subtree with keys k_i, \dots, k_j and intervals d_{i-1}, \dots, d_j must be optimal for the respective sub-problem.³⁹
- Consider all subtrees with roots k_r and optimal subtrees for keys k_i, \dots, k_{r-1} and k_{r+1}, \dots, k_j

³⁹The usual argument: if it was not optimal, it could be replaced by a better solution improving the overall solution.

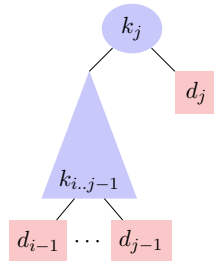
Sub-trees for Searching



empty left subtree



non-empty left and
right subtrees



empty right subtree

Expected Search Costs

Let $\text{depth}_T(k)$ be the depth of a node k in the sub-tree T . Let k be the root of subtrees T_r and T_{L_r} and T_{R_r} be the left and right sub-tree of T_r . Then

$$\text{depth}_T(k_i) = \text{depth}_{T_{L_r}}(k_i) + 1, (i < r)$$

$$\text{depth}_T(k_i) = \text{depth}_{T_{R_r}}(k_i) + 1, (i > r)$$

Expected Search Costs

Let $e[i, j]$ be the costs of an optimal search tree with nodes k_i, \dots, k_j .

Base case $e[i, i - 1]$, expected costs d_{i-1}

Let $w(i, j) = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l$.

If k_r is the root of an optimal search tree with keys k_i, \dots, k_j , then

$$e[i, j] = p_r + (e[i, r - 1] + w(i, r - 1)) + (e[r + 1, j] + w(r + 1, j))$$

with $w(i, j) = w(i, r - 1) + p_r + w(r + 1, j)$:

$$e[i, j] = e[i, r - 1] + e[r + 1, j] + w(i, j).$$

Dynamic Programming

$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1, \\ \min_{i \leq r \leq j} \{e[i, r - 1] + e[r + 1, j] + w[i, j]\} & \text{if } i \leq j \end{cases}$$

Computation

Tables $e[1 \dots n + 1, 0 \dots n]$, $w[1 \dots n + 1, 0 \dots m]$, $r[1 \dots n, 1 \dots n]$ Initially

■ $e[i, i - 1] \leftarrow q_{i-1}$, $w[i, i - 1] \leftarrow q_{i-1}$ for all $1 \leq i \leq n + 1$.

We compute

$$w[i, j] = w[i, j - 1] + p_j + q_j$$

$$e[i, j] = \min_{i \leq r \leq j} \{e[i, r - 1] + e[r + 1, j] + w[i, j]\}$$

$$r[i, j] = \arg \min_{i \leq r \leq j} \{e[i, r - 1] + e[r + 1, j] + w[i, j]\}$$

for intervals $[i, j]$ with increasing lengths $l = 1, \dots, n$, each for $i = 1, \dots, n - l + 1$. Result in $e[1, n]$, reconstruction via r . Runtime $\Theta(n^3)$.

Example

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

e

j						
0	0.05					
1	0.45	0.10				
2	0.90	0.40	0.05			
3	1.25	0.70	0.25	0.05		
4	1.75	1.20	0.60	0.30	0.05	
5	2.75	2.00	1.30	0.90	0.50	0.10
	1	2	3	4	5	6

w

j						
0	0.05					
1	0.30	0.10				
2	0.45	0.25	0.05			
3	0.55	0.35	0.15	0.05		
4	0.70	0.50	0.30	0.20	0.05	
5	1.00	0.80	0.60	0.50	0.35	0.10
	1	2	3	4	5	6

r

j						
1	1					
2	1	2				
3	2	2	3			
4	2	2	4	4		
5	2	4	5	5	5	
	1	2	3	4	5	