Advanced Core in Algorithm Design #6
算法設計要論 第6回

Yasushi Kawase
河瀬 康志

Nov. 8th, 2022

last update: 2:37pm, November 21, 2023

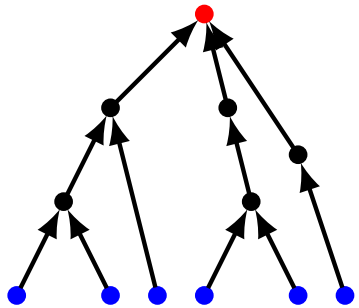
Schedule

Lec. #	Date	Topics
1	10/4	Introduction, Stable matching
2	10/11	Basics of Algorithm Analysis, Greedy Algorithms (1/2)
3	10/18	Greedy Algorithms (2/2)
4	10/25	Divide and Conquer (1/2)
5	11/1	Divide and Conquer (2/2)
6	11/8	Dynamic Programming (1/2)
7	11/15	Dynamic Programming (2/2)
—	11/22	Thursday Classes
8	11/29	Network Flow (1/2)
9	12/6	Network Flow (2/2)
10	12/13	NP and Computational Intractability
11	12/20	Approximation Algorithms (1/2)
12	12/27	Approximation Algorithms (2/2)
13	1/10	Randomized Algorithms

Algorithmic Paradigms

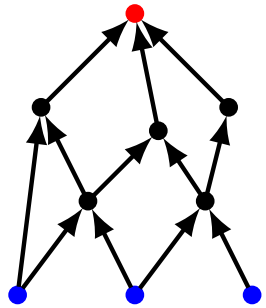
Divide-and-Conquer

- Divide up problem into several independent problems
- Solve each subproblems recursively
- Combine solutions to subproblems into overall solution



Dynamic Programming

- Divide up problem into several overlapping problems
- Solve each subproblems recursively (or bottom up)
- Combine solutions to subproblems into overall solution



Outline

1. Weighted interval scheduling
2. Subset sums and knapsacks
3. Segmented least squares
4. Sequence alignment

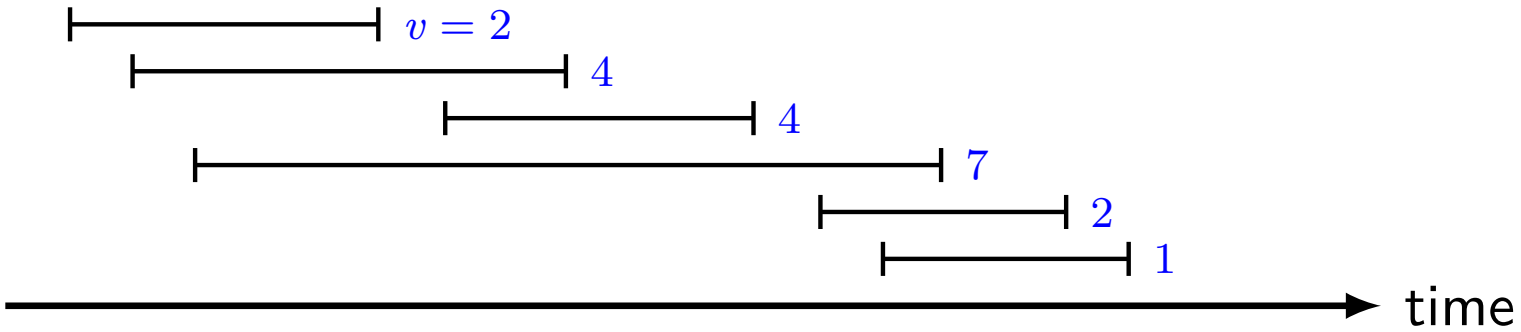
Weighted interval scheduling

Problem

- Input: jobs $J = \{1, 2, \ldots, n\}$,
  job j starts at s_j, finishes at f_j, and has value $v_j > 0$
- Goal: find maximum value subset of mutually compatible jobs

two jobs that don't overlap

Example



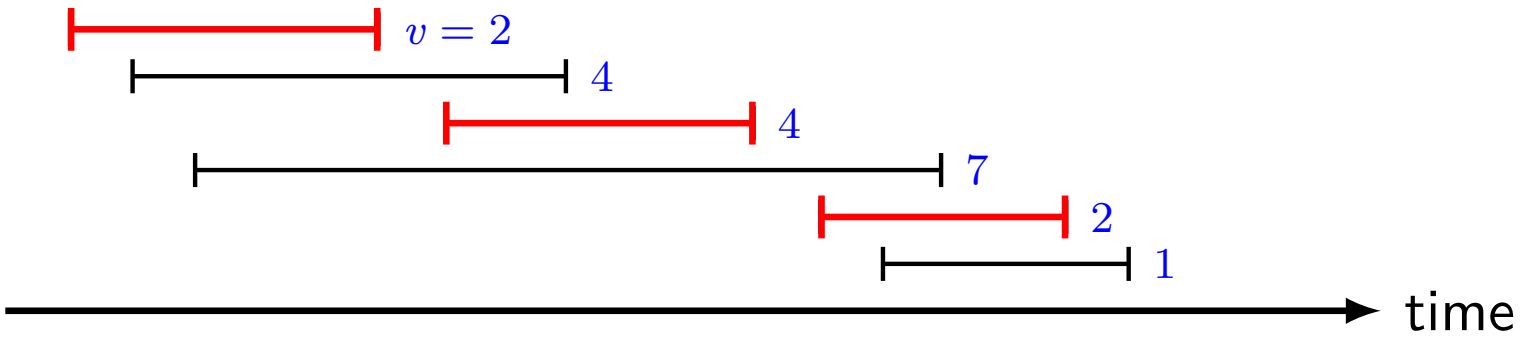
Weighted interval scheduling

Problem

- Input: jobs $J = \{1, 2, \ldots, n\}$,
  job j starts at s_j, finishes at f_j, and has value $v_j > 0$
- Goal: find maximum value subset of mutually compatible jobs

two jobs that don't overlap

Example



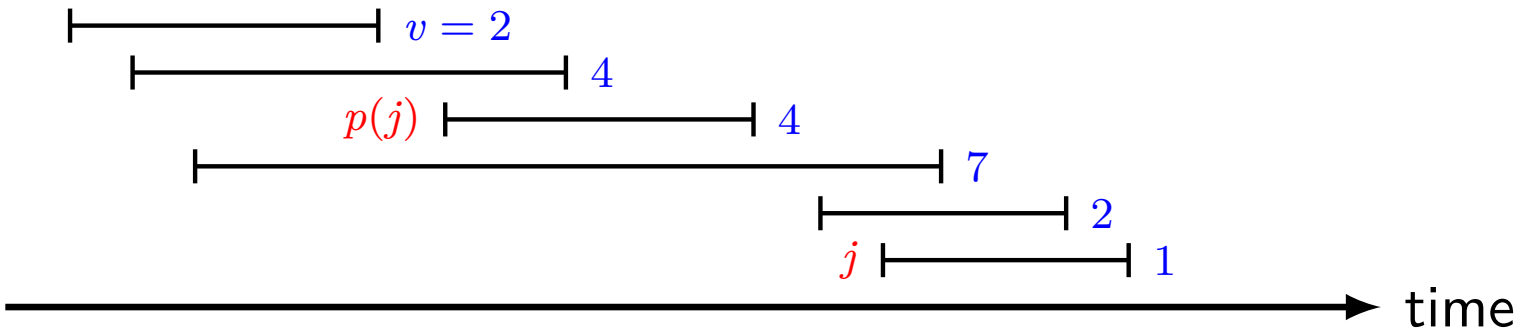
Recursive relation

- sort the jobs by finish time ($f_1 \leq f_2 \leq \cdots \leq f_n$)
- $\mathrm{OPT}(j)$: optimal solution for subproblem consisting only of $\{1, 2, \ldots, j\}$

Recursive formula

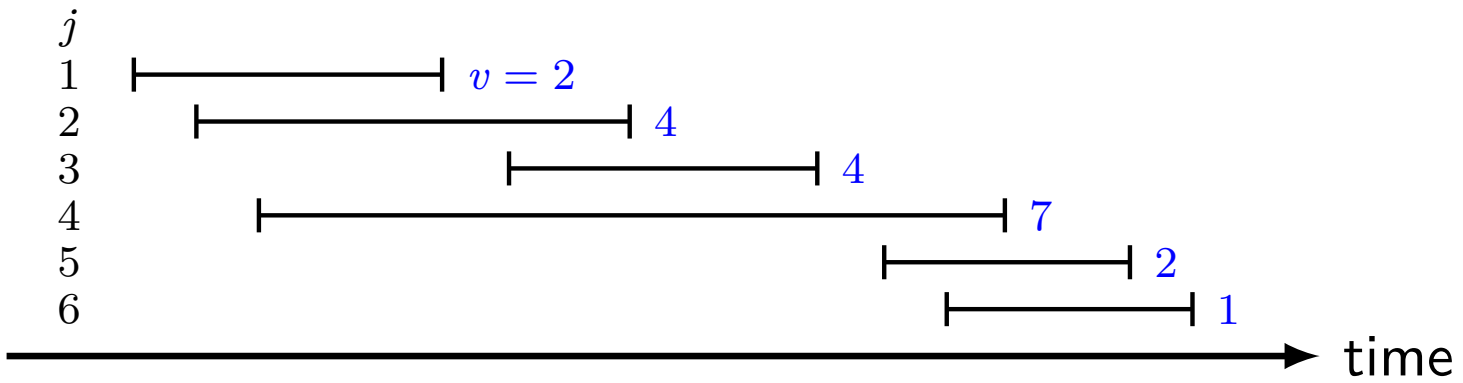
$$\mathrm{OPT}(j) = \begin{cases} 0 & \text{if } j = 0, \\ \max\{v_j + \mathrm{OPT}(p(j)),\ \mathrm{OPT}(j-1)\} & \text{if } j > 0 \end{cases}$$

maximum i such that $f(i) \leq s(j)$ (found in $\mathrm{O}(\log n)$ via binary search)



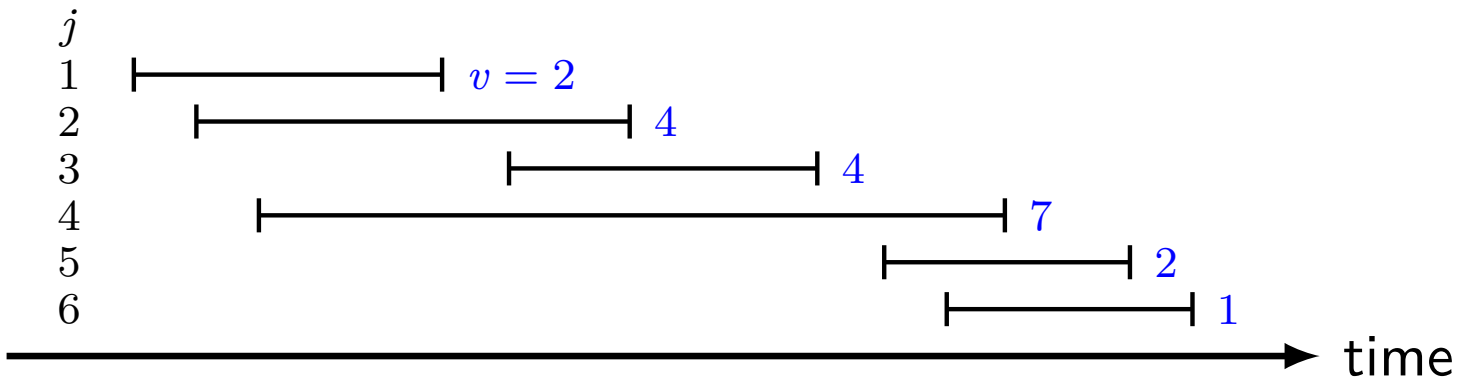
Time complexity: $\mathrm{O}(n \log n)$

Example



j	0	1	2	3	4	5	6
$\mathrm{OPT}(j)$							

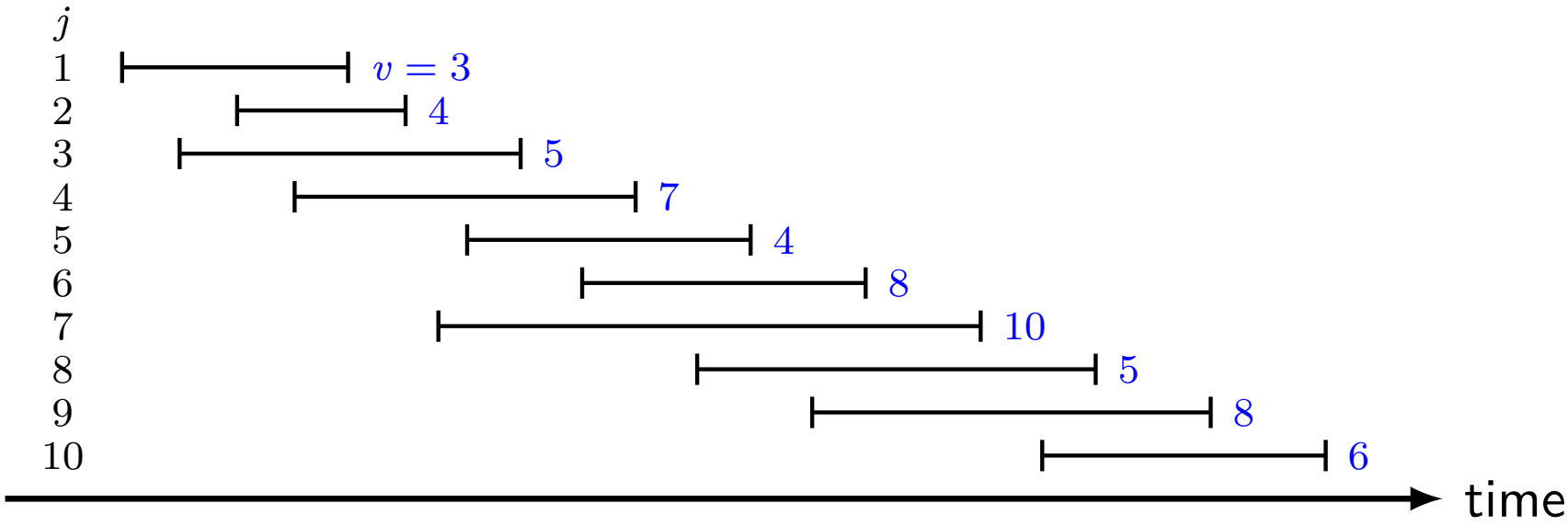
Example



j	0	1	2	3	4	5	6
$\mathrm{OPT}(j)$	0	2	4	6	7	8	8

Quiz

What is the optimal value of the following problem?



Outline

1. Weighted interval scheduling
2. Subset sums and knapsacks
3. Segmented least squares
4. Sequence alignment

Subset sum

Problem

- Input: positive integers $a_1, a_2, \ldots, a_n$ and W
- Goal: find $I \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in I} a_i = W$

Examples

- $a = (1, 3, 7, 8, 11)$, $W = 20$ → possible $(1 + 8 + 11 = 20)$
- $a = (2, 4, 8, 10, 12)$, $W = 19$ → impossible

Dynamic programming for subset sum problem

$\psi(k, w)$: solution for $a_1, a_2, \ldots, a_k$ and w (`True` or `False`)

Recursive formula

$$\psi(k, w) = \begin{cases} \texttt{True} & \text{if } k = 0 \text{ and } w = 0 \\ \texttt{False} & \text{if } k = 0 \text{ and } w > 0 \\ \psi(k-1, w) & \text{if } w < a_k \\ \psi(k-1, w) \vee \psi(k-1, w - a_k) & \text{otherwise} \end{cases}$$

Compute for $k = 0, 1, \ldots, n$ and $w = 0, 1, \ldots, W$ → $O(nW)$ time

Example: $a_1 = 2, a_2 = 4, a_3 = 3,\ W = 5$

	$k \setminus w$	0	1	2	3	4	5
	0	—	—	—	—	—	—
$a_1 = 2$	1	—	—	—	—	—	—
$a_2 = 4$	2	—	—	—	—	—	—
$a_3 = 3$	3	—	—	—	—	—	—

Dynamic programming for subset sum problem

$\psi(k, w)$: solution for $a_1, a_2, \ldots, a_k$ and w (`True` or `False`)

Recursive formula

$$\psi(k, w) = \begin{cases} \texttt{True} & \text{if } k = 0 \text{ and } w = 0 \\ \texttt{False} & \text{if } k = 0 \text{ and } w > 0 \\ \psi(k-1, w) & \text{if } w < a_k \\ \psi(k-1, w) \vee \psi(k-1, w - a_k) & \text{otherwise} \end{cases}$$

Compute for $k = 0, 1, \ldots, n$ and $w = 0, 1, \ldots, W$ → $O(nW)$ time

Example: $a_1 = 2, a_2 = 4, a_3 = 3,\ W = 5$

	$k \setminus w$	0	1	2	3	4	5
	0	`True`	`False`	`False`	`False`	`False`	`False`
$a_1 = 2$	1	—	—	—	—	—	—
$a_2 = 4$	2	—	—	—	—	—	—
$a_3 = 3$	3	—	—	—	—	—	—

Dynamic programming for subset sum problem

$\psi(k, w)$: solution for $a_1, a_2, \ldots, a_k$ and w (`True` or `False`)

Recursive formula

$$\psi(k, w) = \begin{cases} \texttt{True} & \text{if } k = 0 \text{ and } w = 0 \\ \texttt{False} & \text{if } k = 0 \text{ and } w > 0 \\ \psi(k-1, w) & \text{if } w < a_k \\ \psi(k-1, w) \vee \psi(k-1, w - a_k) & \text{otherwise} \end{cases}$$

Compute for $k = 0, 1, \ldots, n$ and $w = 0, 1, \ldots, W$ → $O(nW)$ time

Example: $a_1 = 2, a_2 = 4, a_3 = 3,\ W = 5$

	$k \setminus w$	0	1	2	3	4	5
	0	`True`	`False`	`False`	`False`	`False`	`False`
$a_1 = 2$	1	`True`	`False`	`True`	`False`	`False`	`False`
$a_2 = 4$	2	—	—	—	—	—	—
$a_3 = 3$	3	—	—	—	—	—	—

Dynamic programming for subset sum problem

$\psi(k, w)$: solution for $a_1, a_2, \ldots, a_k$ and w (`True` or `False`)

Recursive formula

$$\psi(k, w) = \begin{cases} \texttt{True} & \text{if } k = 0 \text{ and } w = 0 \\ \texttt{False} & \text{if } k = 0 \text{ and } w > 0 \\ \psi(k-1, w) & \text{if } w < a_k \\ \psi(k-1, w) \vee \psi(k-1, w - a_k) & \text{otherwise} \end{cases}$$

Compute for $k = 0, 1, \ldots, n$ and $w = 0, 1, \ldots, W$ → $O(nW)$ time

Example: $a_1 = 2, a_2 = 4, a_3 = 3,\ W = 5$

	$k \setminus w$	0	1	2	3	4	5
	0	True	False	False	False	False	False
$a_1 = 2$	1	True	False	True	False	False	False
$a_2 = 4$	2	True	False	True	False	True	False
$a_3 = 3$	3	—	—	—	—	—	—

Dynamic programming for subset sum problem

$\psi(k, w)$: solution for $a_1, a_2, \ldots, a_k$ and w (`True` or `False`)

Recursive formula

$$\psi(k, w) = \begin{cases} \texttt{True} & \text{if } k = 0 \text{ and } w = 0 \\ \texttt{False} & \text{if } k = 0 \text{ and } w > 0 \\ \psi(k-1, w) & \text{if } w < a_k \\ \psi(k-1, w) \vee \psi(k-1, w - a_k) & \text{otherwise} \end{cases}$$

Compute for $k = 0, 1, \ldots, n$ and $w = 0, 1, \ldots, W$ → $O(nW)$ time

Example: $a_1 = 2, a_2 = 4, a_3 = 3,\ W = 5$

	$k \setminus w$	0	1	2	3	4	5
	0	True	False	False	False	False	False
$a_1 = 2$	1	True	False	True	False	False	False
$a_2 = 4$	2	True	False	True	False	True	False
$a_3 = 3$	3	True	False	True	True	True	True

Knpsack problem

Problem

- Input: items $E = \{1, 2, \ldots, n\}$ and a capacity $W \in \mathbb{Z}_+$
  item i has value $v_i \in \mathbb{Z}_+$ and size $w_i \in \mathbb{Z}_+$
- Goal: maximize $\sum_{i \in I} v_i$ subject to $I \subseteq \{1, 2, \ldots, n\}$ $\sum_{i \in I} w_i \leq W$

Examples

$W = 16$

i	v_i	w_i
1	55	4
2	61	2
3	82	9
4	38	1
5	63	3

Knpsack problem

Problem

- Input: items $E = \{1, 2, \ldots, n\}$ and a capacity $W \in \mathbb{Z}_+$
  item i has value $v_i \in \mathbb{Z}_+$ and size $w_i \in \mathbb{Z}_+$
- Goal: maximize $\sum_{i \in I} v_i$ subject to $I \subseteq \{1, 2, \ldots, n\}$ $\sum_{i \in I} w_i \leq W$

Examples

$W = 16$

i	v_i	w_i
1	55	4
2	61	2
3	82	9
4	38	1
5	63	3

optimal value is $61 + 82 + 38 + 63 = 244$

Dynamic programming for knapsack problem

$$\mathrm{OPT}(k, w) = \max\{\textstyle\sum_{i \in X} v_i \mid X \subseteq \{1, 2, \ldots, k\},\ \sum_{i \in X} w_i \leq w\}$$

Recursive formula

$$\mathrm{OPT}(k, w) = \begin{cases} 0 & \text{if } k = 0, \\ \mathrm{OPT}(k-1, w) & \text{if } w_k > w \\ \max\{\mathrm{OPT}(k-1, w),\ \mathrm{OPT}(k-1, w-w_k) + v_k\} & \text{otherwise} \end{cases}$$

Compute for $k = 0, 1, \ldots, n$ and $w = 0, 1, \ldots, W$ → $\mathrm{O}(nW)$ time

Example $(w_i, v_i) = (4, 55), (2, 61), (9, 82), (1, 38), (3, 63),\ W = 16$

$k \backslash w$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	55	55	55	55	55	55	55	55	55	55	55	55	55
2	0	0	61	61	61	61	116	116	116	116	116	116	116	116	116	116	116
3	0	0	61	61	61	61	116	116	116	116	116	143	143	143	143	198	198
4	0	38	61	99	99	99	116	154	154	154	154	154	181	181	181	198	236
5	0	38	61	99	101	124	162	162	162	179	217	217	217	217	217	244	244

Outline

1 Weighted interval scheduling

2 Subset sums and knapsacks

3 Segmented least squares

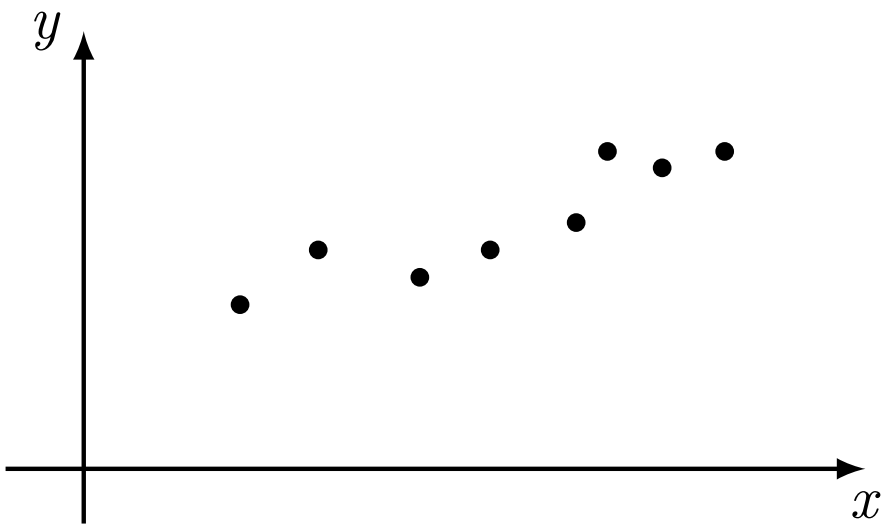
4 Sequence alignment

Least squares

Problem

- Input: n points $p_1, p_2, \ldots, p_i, \ldots, p_n \in \mathbb{R}^2$ ((x_i, y_i))
- Goal: find a line $y = ax + b$ that minimizes the sum of the squared error ($\sum_{i=1}^{n}(y_i - ax_i - b)^2$)

Example



Least squares

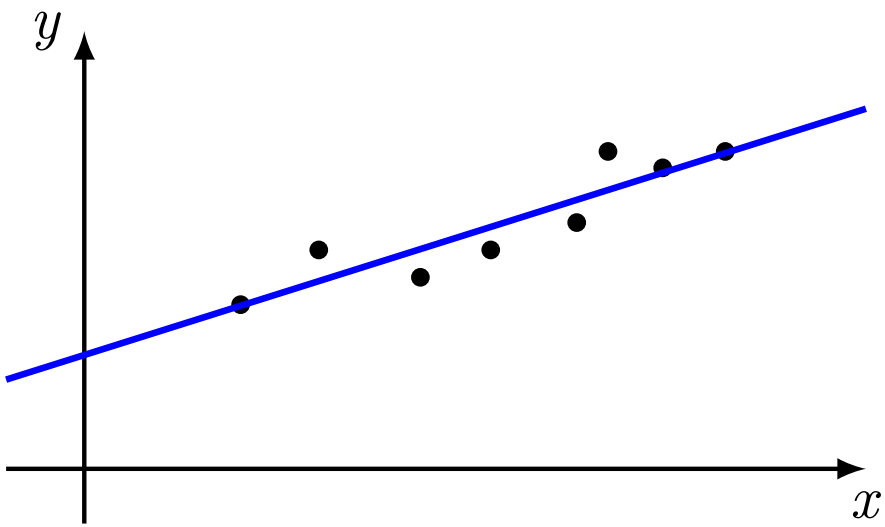
Problem

(x_i, y_i)

- Input: n points $p_1, p_2, \ldots, p_i, \ldots, p_n \in \mathbb{R}^2$
- Goal: find a line $y = ax + b$ that minimizes the sum of the squared error

$\sum_{i=1}^{n}(y_i - ax_i - b)^2$

Example



Least squares

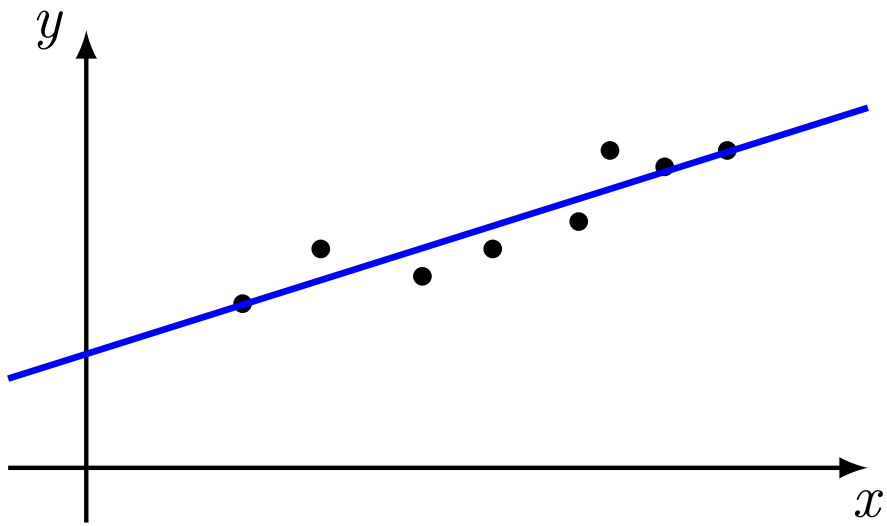
Problem

(x_i, y_i)

- Input: n points $p_1, p_2, \ldots, p_i, \ldots, p_n \in \mathbb{R}^2$
- Goal: find a line $y = ax + b$ that minimizes the sum of the squared error

$\sum_{i=1}^{n}(y_i - ax_i - b)^2$

Example



Optimal solution: $a = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$, $b = \frac{\sum_{i=1}^{n} y_i - a\sum_{i=1}^{n} x_i}{n}$

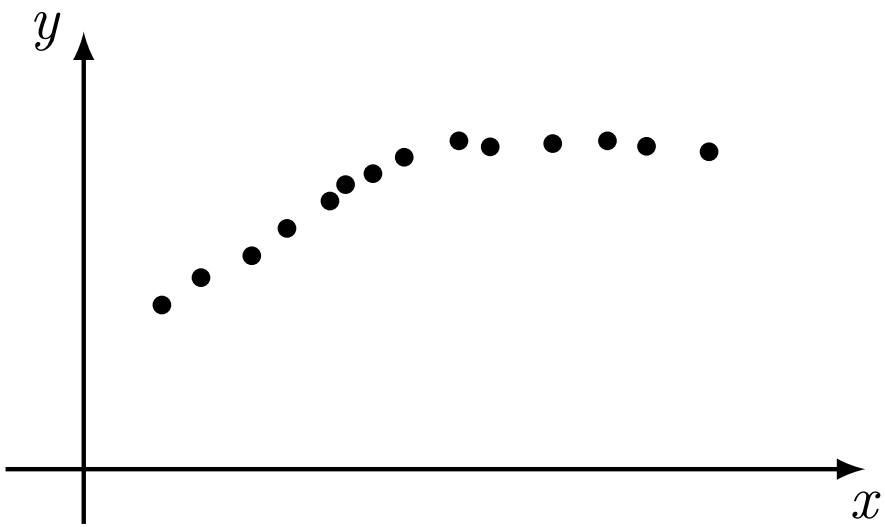
→ $O(n)$ time

Segmented least squares

Problem

- Input: n points $p_1, p_2, \ldots, p_i, \ldots, p_n \in \mathbb{R}^2$ and $c > 0$ ((x_i, y_i))
- Goal: find segments $y = f(x)$ that minimizes $\sum_{i=1}^{n}(y_i - f(x_i))^2 + cL$ (L: # of segements)

Examples

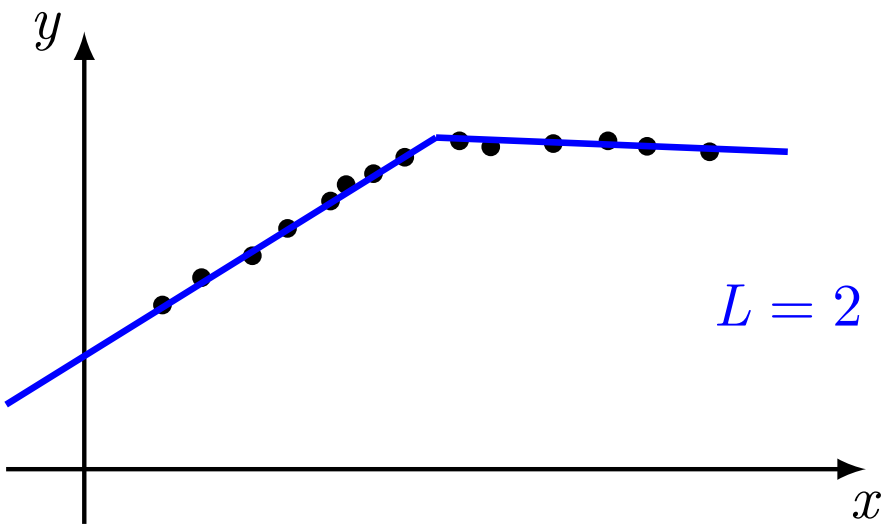


Segmented least squares

Problem

- Input: n points $p_1, p_2, \ldots, p_i, \ldots, p_n \in \mathbb{R}^2$ and $c > 0$ ((x_i, y_i))
- Goal: find segments $y = f(x)$ that minimizes $\sum_{i=1}^{n}(y_i - f(x_i))^2 + cL$ (L: # of segements)

Examples



Dynamic programming

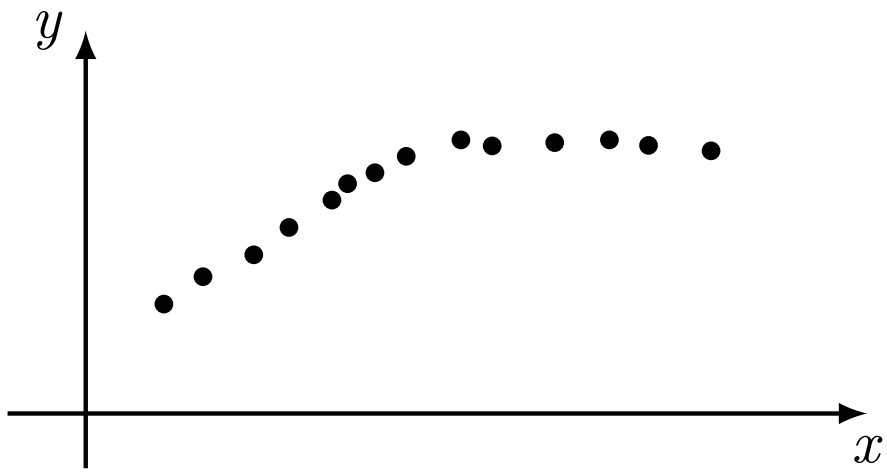
Notation

- $\mathrm{OPT}(j)$: opt. val. of segmented least squares for $p_1, p_2, \ldots, p_j$
- e_{ij}: opt. val. of least squares for $p_i, p_{i+1}, \ldots, p_j$ computable in $\mathrm{O}(n)$ time

Recursive formula

$$\mathrm{OPT}(j) = \begin{cases} 0 & \text{if } j = 0 \\ \min_{1 \le i \le j}\{e_{ij} + c + \mathrm{OPT}(i-1)\} & \text{if } j > 0 \end{cases}$$

Compute for $j = 0, 1, \ldots, n$, each of which takes $\mathrm{O}(n^2)$ → $\mathrm{O}(n^3)$ time



Improve the computational time

Bottleneck: computing $e_{ij} = \sum_{k=i}^{j}(y_k - a_{ij}x_k - b_{ij})^2$ takes $\mathrm{O}(n)$ time

- $a_{ij} = \frac{(j-i+1)\sum_{k=i}^{j} x_k y_k - (\sum_{k=i}^{j} x_k)(\sum_{k=i}^{j} y_k)}{(j-i+1)\sum_{k=i}^{j} x_k^2 - \left(\sum_{k=i}^{j} x_k\right)^2}$
- $b_{ij} = \frac{\sum_{k=i}^{j} y_k - a_{ij}\sum_{k=i}^{j} x_k}{j-i+1}$

Approach

- precompute $\sum_{k=1}^{j} x_k$, $\sum_{k=1}^{j} y_k$, $\sum_{k=1}^{j} x_k^2$, $\sum_{k=1}^{j} x_k y_k$ $(\forall j)$ in $\mathrm{O}(n)$ time
- using cumulative sums, e_{ij} can be computed in $\mathrm{O}(1)$ time

→ segmented least squares is solvable in $\mathrm{O}(n^2)$ time

Outline

1. Weighted interval scheduling
2. Subset sums and knapsacks
3. Segmented least squares
4. Sequence alignment

String similarity

Q: How similar are two strings $X = x_1 x_2 \dots x_m$ and $Y = y_1 y_2 \dots y_n$

Alignment of X and Y

set of ordered pairs x_i–y_j such that each character appears in at most one pair and no crossing

Example "ocurrance" and "occurrence"

o	c	u	r	r	a	n	c	e	–
o	c	c	u	r	r	e	n	c	e

6 mismatches, 1 gap

o	c	–	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

1 mismatch, 1 gap

o	c	–	u	r	r	–	a	n	c	e
o	c	c	u	r	r	e	–	n	c	e

0 mismatches, 3 gaps

Edit distance

Edit distance: minimum sum of gap and mismatch penalties

- Gap penalty $\delta \geq 0$
- Mismatch penalty $\alpha_{pq} \geq 0 \quad (\alpha_{pp} = 0)$

Example

o	c	−	u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

$\text{cost} = \delta + \alpha_{ae}$

o	c	−	u	r	r	−	a	n	c	e
o	c	c	u	r	r	e	−	n	c	e

$\text{cost} = 3\delta$

Problem

- Input: two strings $X = x_1 x_2 \ldots x_m$ and $Y = y_1 y_2 \ldots y_n$, penalty δ, α_{pq}
- Goal: compute the edit distance

Dynamic programming for knapsack problem

$\mathrm{OPT}(i, j)$: edit distance for $x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j$

Recursive formula

$$\mathrm{OPT}(i,j) = \begin{cases} j\delta & \text{if } i = 0 \\ i\delta & \text{if } j = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + \mathrm{OPT}(i-1, j-1) \\ \delta + \mathrm{OPT}(i-1, j) \\ \delta + \mathrm{OPT}(i, j-1) \end{cases} & \text{otherwise} \end{cases}$$

Compute for $i = 0, 1, \ldots, m$ and $j = 0, 1, \ldots, n$ → $\mathrm{O}(mn)$ time

X:

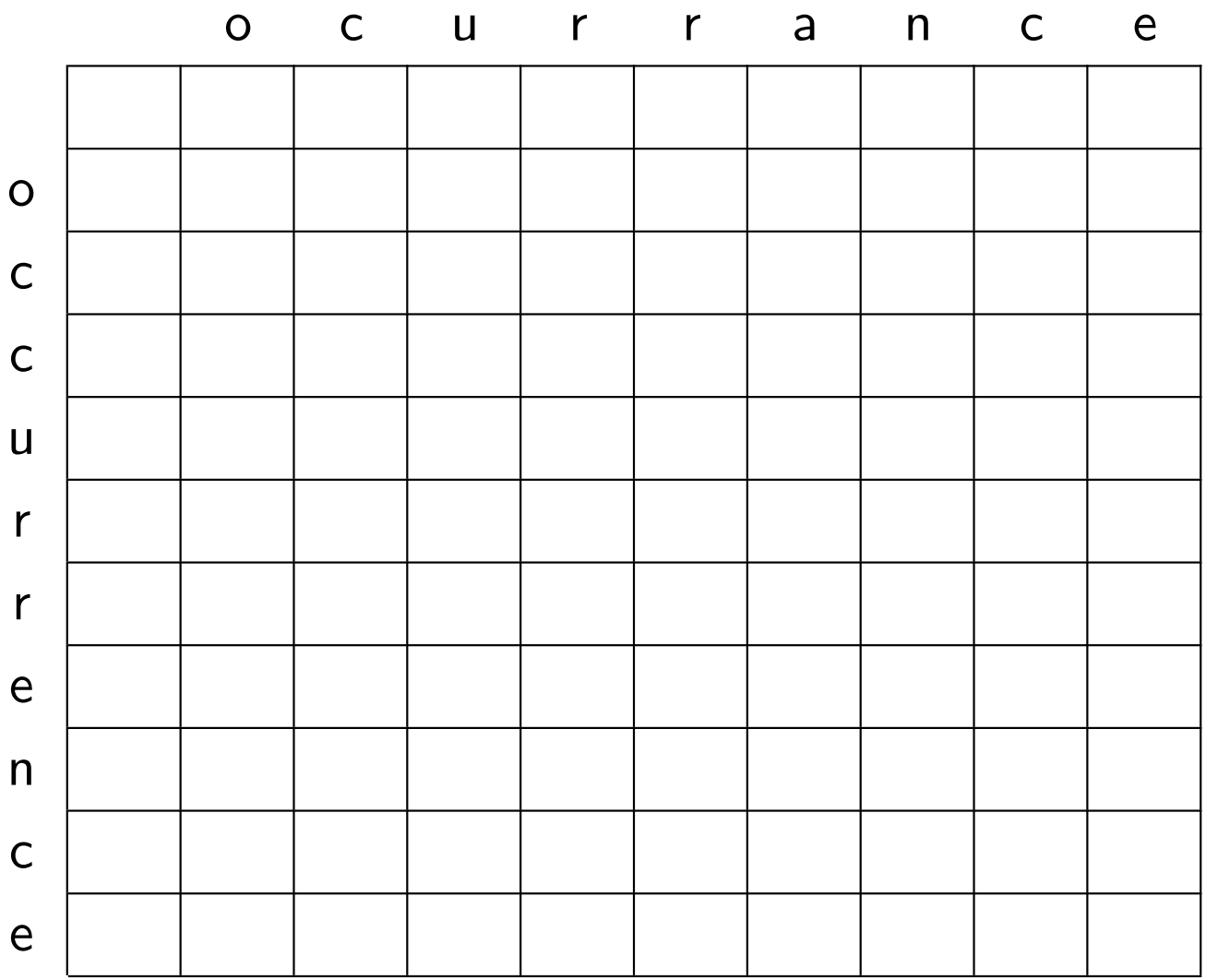
o	c	u	r	r	a	n	c	e

Y:

o	c	c	u	r	r	e	n	c	e

Example

$\delta = 2$, $\alpha_{pq} = 1$ (for any distinct p, q)



Example

$\delta = 2$, $\alpha_{pq} = 1$ (for any distinct p, q)

		o	c	u	r	r	a	n	c	e
	0	2	4	6	8	10	12	14	16	18
o	2	0	2	4	6	8	10	12	14	16
c	4	2	0	2	4	6	8	10	12	14
c	6	4	2	1	3	5	7	9	10	12
u	8	6	4	2	2	4	6	8	10	11
r	10	8	6	4	2	2	4	6	8	10
r	12	10	8	6	4	2	3	5	7	9
e	14	12	10	8	6	4	3	4	6	7
n	16	14	12	10	8	6	5	3	5	7
c	18	16	14	12	10	8	7	5	3	5
e	20	18	16	14	12	10	9	7	5	3

Quiz

What is the edit distance if $\delta = 3$, $\alpha_{pq} = 2$ (for any distinct p, q)

		o	c	u	r	r	a	n	c	e
o										
c										
c										
u										
r										
r										
e										
n										
c										
e										