CS 466
Introduction to Bioinformatics

Lecture 5

Mohammed El-Kebir

September 9, 2020

Course Announcements

Instructor:

- Mohammed El-Kebir (melkebir)
- Office hours: Wednesdays, 3:15-4:15pm

TA:

- Mondays, 3-4pm
- Fridays, 9-10am

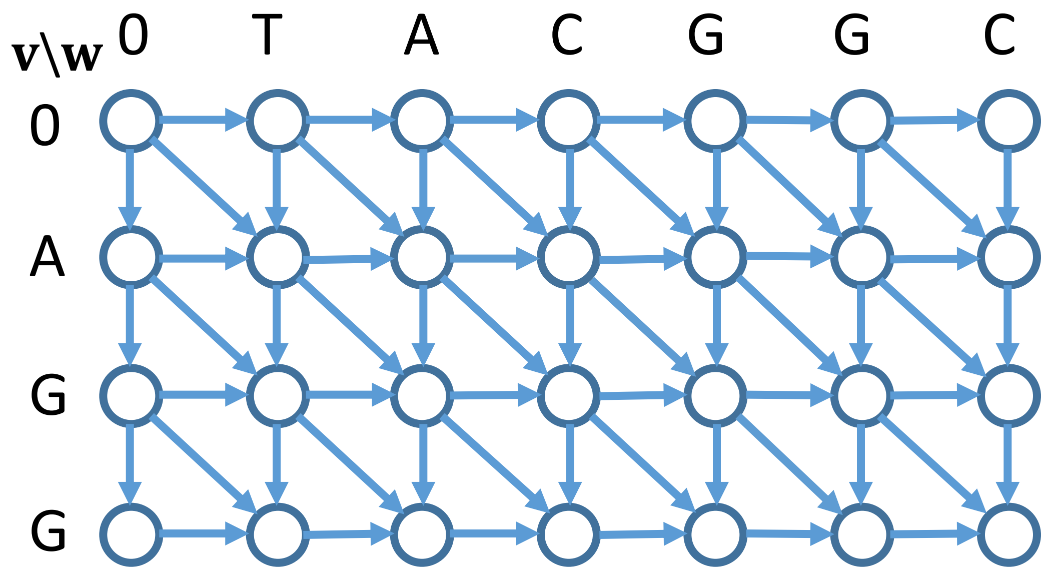
Homework 1: Due on Sept. 17 (11:59pm)

Global, Fitting and Local Alignment

Global Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find alignment of $\mathbf{v}$ and $\mathbf{w}$ with maximum score.
[Needleman-Wunsch algorithm]

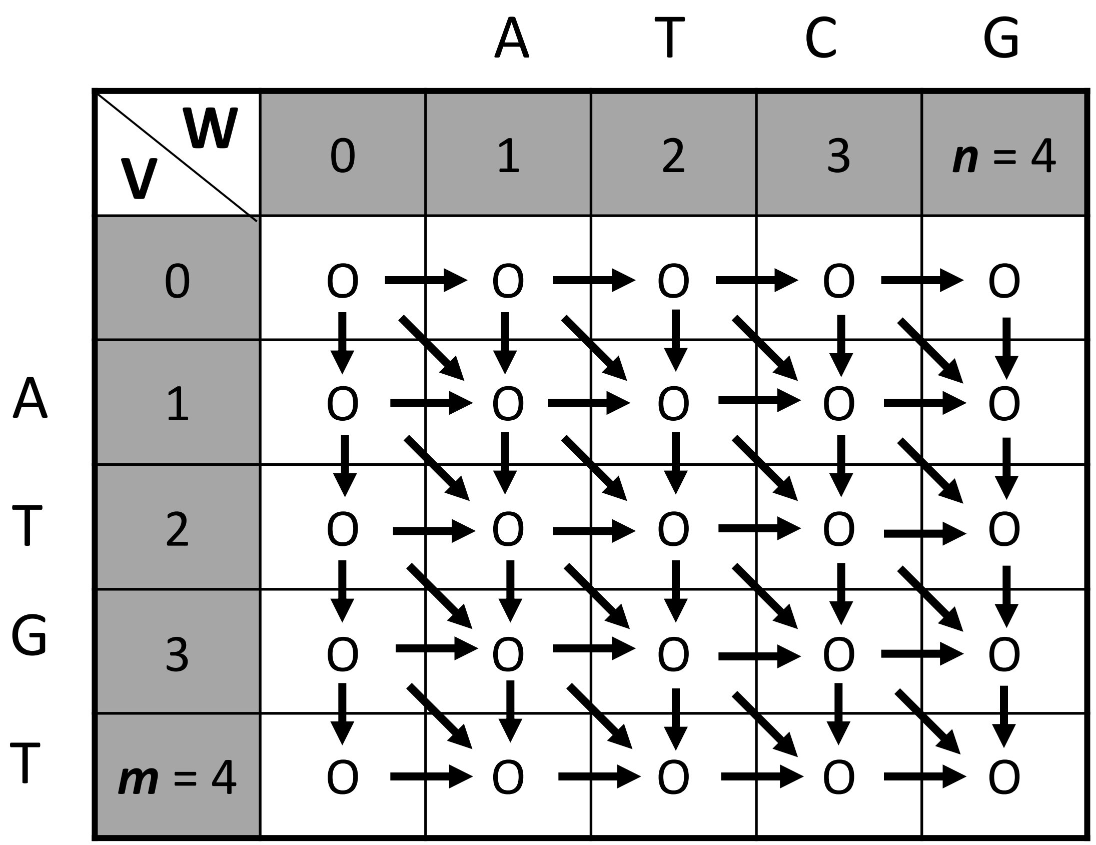
Fitting Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find an alignment of $\mathbf{v}$ and a substring of $\mathbf{w}$ with maximum global alignment score s^* among *all* global alignments of $\mathbf{v}$ and *all* substrings of $\mathbf{w}$

Local Alignment problem: Given strings $\mathbf{v} \in \Sigma^m$ and $\mathbf{w} \in \Sigma^n$ and scoring function δ, find a substring of $\mathbf{v}$ and a substring of $\mathbf{w}$ whose alignment has maximum global alignment score s^* among *all* global alignments of *all* substrings of $\mathbf{v}$ and $\mathbf{w}$
[Smith-Waterman algorithm]



Question: How to assess resulting algorithms?

Time Complexity

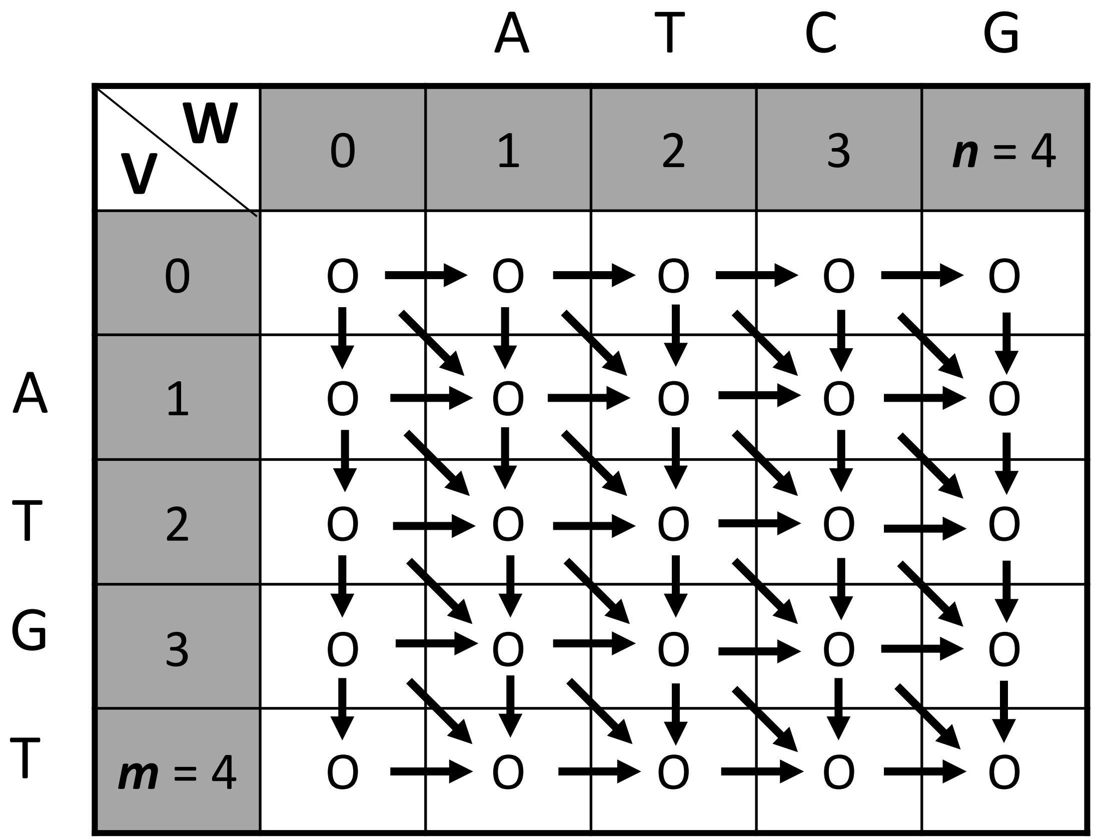


Edit graph is a weighed, directed grid graph $G = (V, E)$ with source vertex $(0, 0)$ and target vertex (m, n). Each edge $((i, j), (k, l))$ has weight depending on direction.

Alignment is a path from source $(0, 0)$ to target (m, n) in edit graph

Running time is $O(mn)$ [**quadratic time**]

Time Complexity



Edit graph is a weighed, directed grid graph $G = (V, E)$ with source vertex $(0, 0)$ and target vertex (m, n). Each edge $((i, j), (k, l))$ has weight depending on direction.

Alignment is a path from source $(0, 0)$ to target (m, n) in edit graph

Running time is $O(mn)$ [**quadratic time**]

Question: Compute alignment faster than $O(mn)$ time? [**subquadratic time**]

Space Complexity

			A	T	C	G
	V \ **W**	0	1	2	3	***n*** = 4
	0					
A	1					
T	2					
G	3					
T	***m*** = 4					

Size of DP table is $(m+1) \times (n+1)$

Thus, space complexity is $O(mn)$
[**quadratic space**]

Example:
To align a short read ($m = 100$) to human genome ($n = 3 \cdot 10^9$), we need 300 GB memory.

Space Complexity

			A	T	C	G
	V \ **W**	0	1	2	3	***n*** = 4
	0					
A	1					
T	2					
G	3					
T	***m*** = 4					

Size of DP table is $(m+1) \times (n+1)$

Thus, space complexity is $O(mn)$
[**quadratic space**]

Example:
To align a short read ($m = 100$) to human genome ($n = 3 \cdot 10^9$), we need 300 GB memory.

Question: How long is an alignment?

$2(m + n)$

$O(m+n)$

Space Complexity

			A	T	C	G
	V \ **W**	0	1	2	3	***n*** = 4
	0					
A	1					
T	2					
G	3					
T	***m*** = 4					

Size of DP table is $(m+1) \times (n+1)$

Thus, space complexity is $O(mn)$
[**quadratic space**]

Example:
To align a short read ($m = 100$) to human genome ($n = 3 \cdot 10^9$), we need 300 GB memory.

Question: How long is an alignment?

Question: Compute alignment in $O(m)$ space? [**linear space**]

Outline

1. Recap of global, fitting, local and gapped alignment
2. Space-efficient alignment
3. Subquadratic time alignment

Reading:

- Jones and Pevzner. Chapters 7.1-7.4
- Lecture notes

Space Efficient Alignment

Computing $s[i,j]$ requires access to:
$s[i-1,j], s[i,j-1]$ and $s[i-1,j-1]$

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, \\ s[i,j-1] + \delta(-, w_j), & \text{if } j > 0, \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

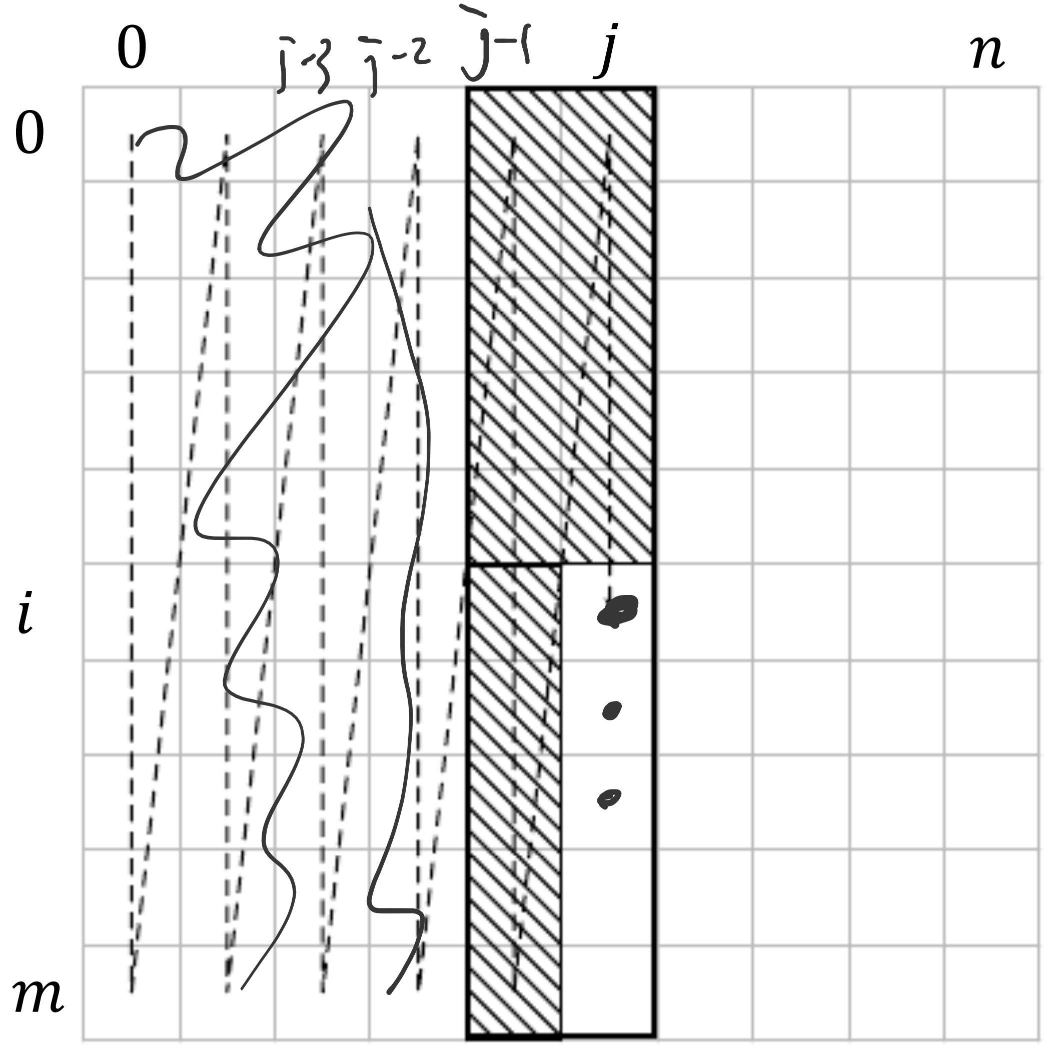


Figure 7.2 Calculating an alignment *score* requires no more than $2n$ space for an $n \times n$ alignment problem. Computing the alignment scores in each column requires only the scores in the preceding column. We show here the dynamic programming array–the data structure that holds the score at each vertex—instead of the graph.

Space Efficient Alignment

Computing $s[i,j]$ requires access to:
$s[i-1,j], s[i,j-1]$ and $s[i-1,j-1]$

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, \\ s[i,j-1] + \delta(-, w_j), & \text{if } j > 0, \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

Thus it suffices to store only two columns to compute optimal alignment score $s[m,n]$, i.e., $2(m+1) = O(m)$ space.

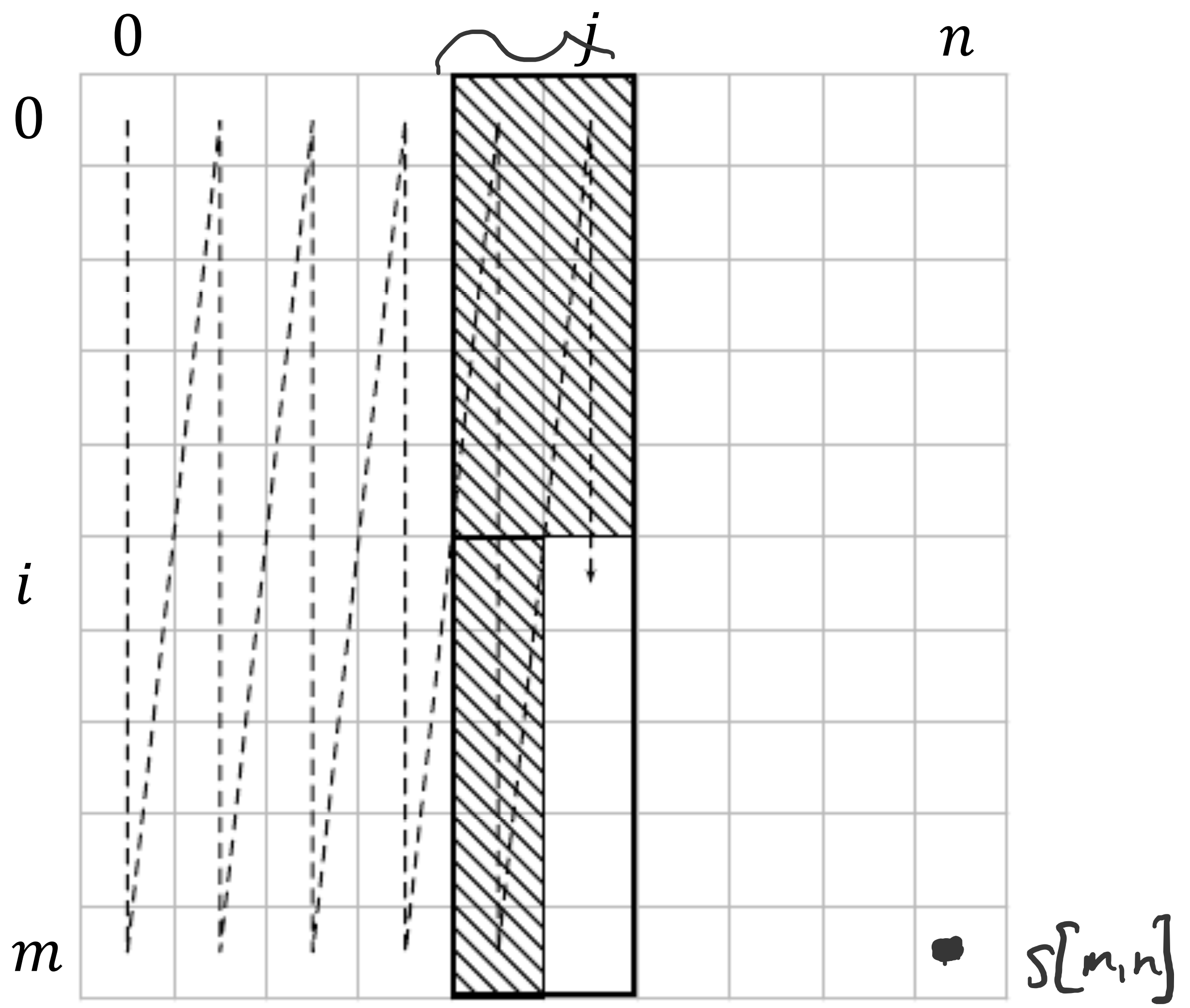


Figure 7.2 Calculating an alignment *score* requires no more than $2n$ space for an $n \times n$ alignment problem. Computing the alignment scores in each column requires only the scores in the preceding column. We show here the dynamic programming array–the data structure that holds the score at each vertex—instead of the graph.

Space Efficient Alignment

O(m) space
O(mj) n time

Computing $s[i,j]$ requires access to:
$s[i-1,j], s[i,j-1]$ and $s[i-1,j-1]$

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] + \delta(v_i, -), & \text{if } i > 0, \\ s[i,j-1] + \delta(-, w_j), & \text{if } j > 0, \\ s[i-1,j-1] + \delta(v_i, w_j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

Thus it suffices to store only two columns to compute optimal alignment score $s[m,n]$, i.e., $2(m+1) = O(m)$ space.

Question: What if we want alignment itself?

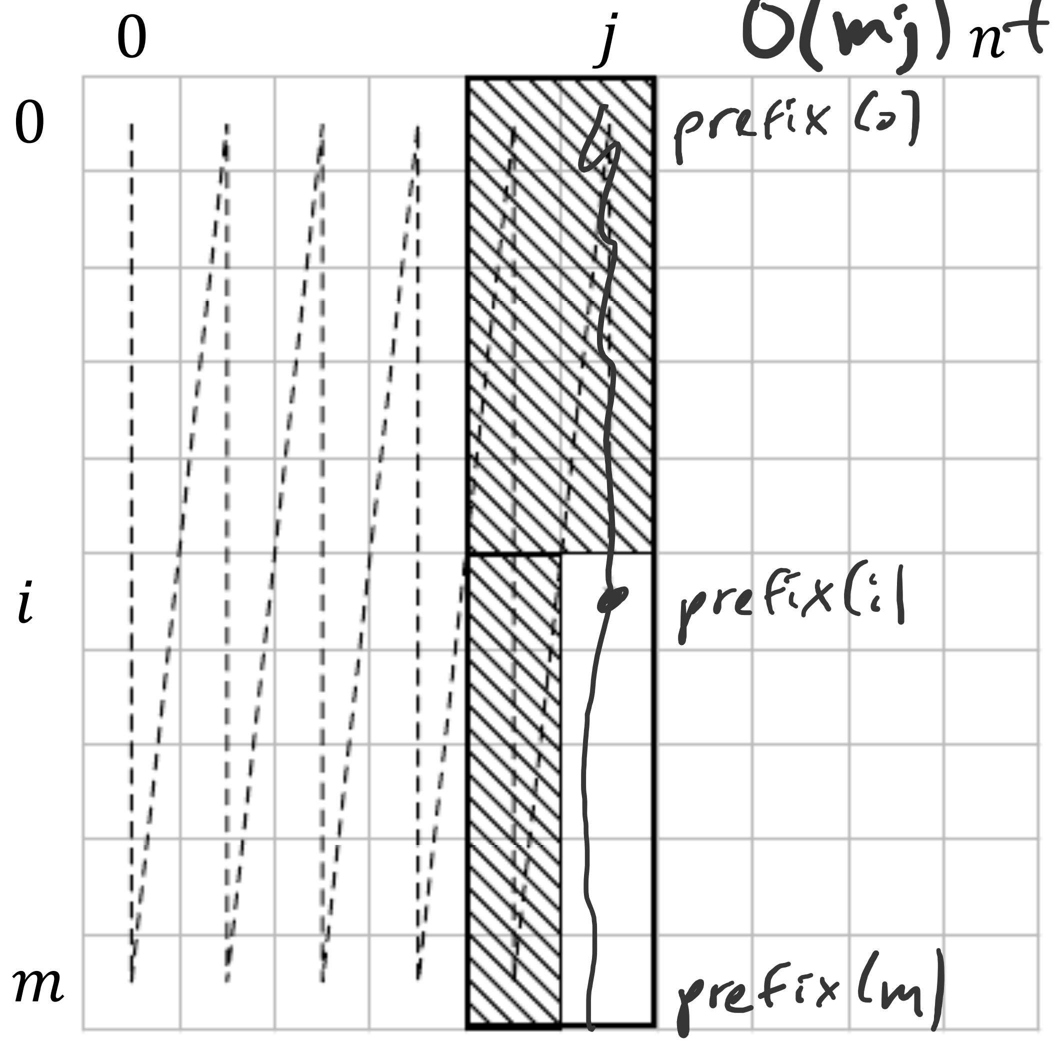
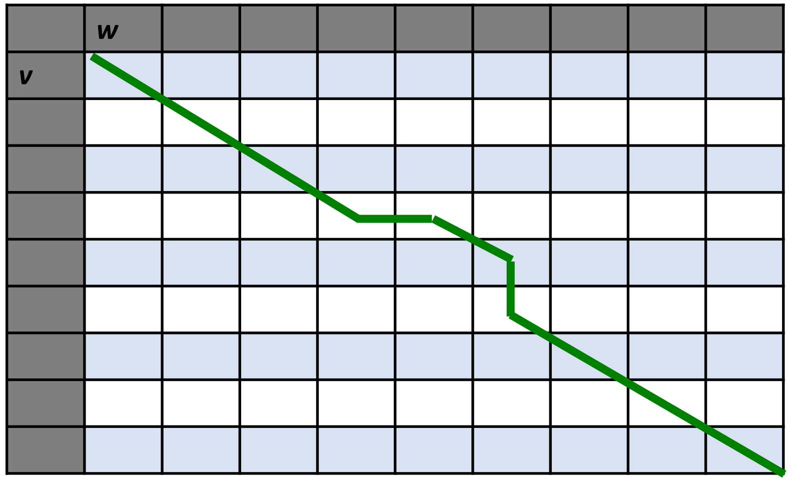
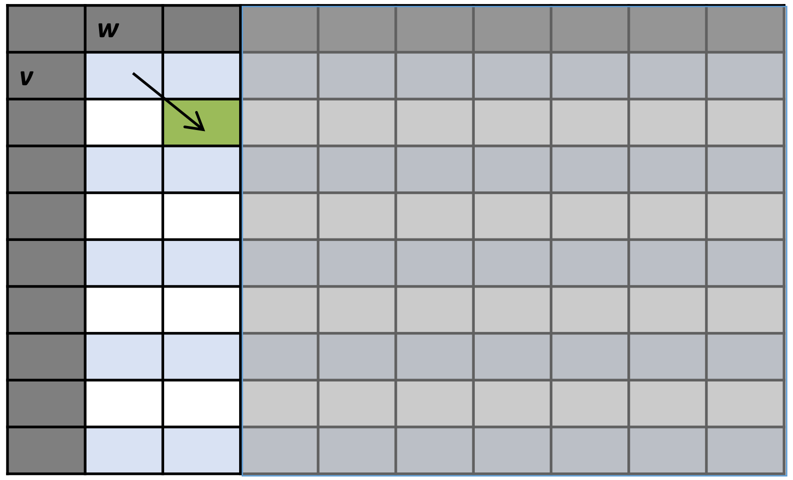


Figure 7.2 Calculating an alignment *score* requires no more than $2n$ space for an $n \times n$ alignment problem. Computing the alignment scores in each column requires only the scores in the preceding column. We show here the dynamic programming array–the data structure that holds the score at each vertex—instead of the graph.

Space Efficient Alignment – First Attempt

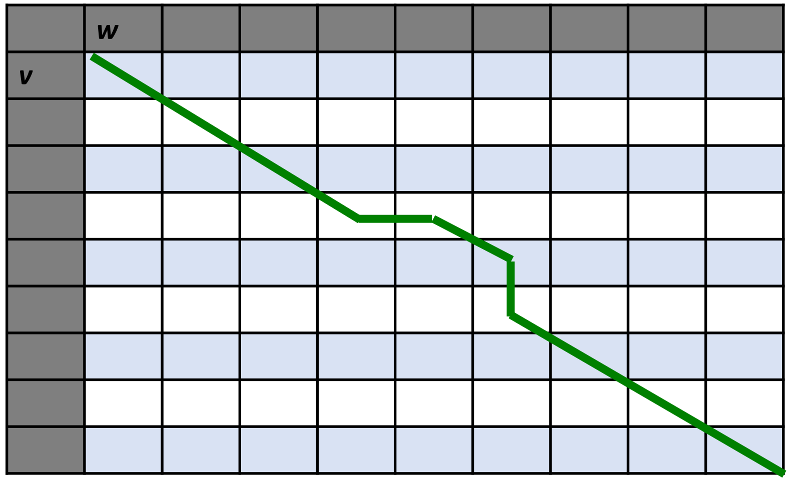
- What if also want optimal alignment?
- **Easy**: keep best pointers as fill in table.
- **No!** Do not know which path to keep until computing recurrence at each step.

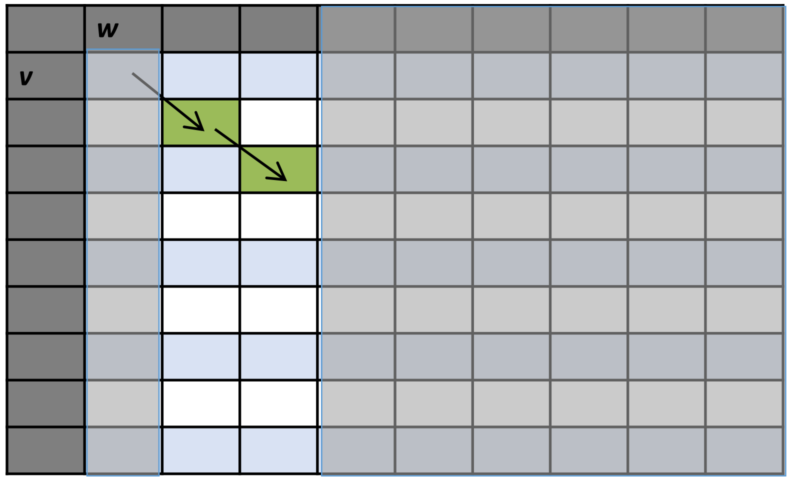




Space Efficient Alignment – First Attempt

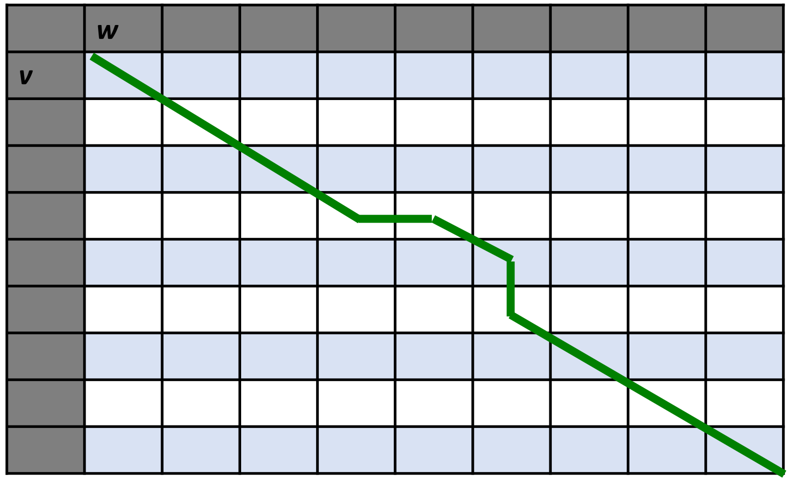
- What if also want optimal alignment?
- **Easy**: keep best pointers as fill in table.
- **No!** Do not know which path to keep until computing recurrence at each step.

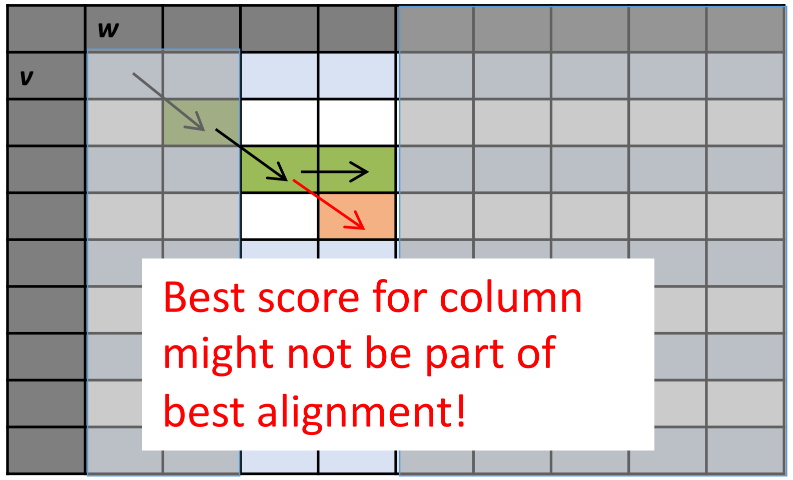




Space Efficient Alignment – First Attempt

- What if also want optimal alignment?
- **Easy**: keep best pointers as fill in table.
- **No!** Do not know which path to keep until computing recurrence at each step.



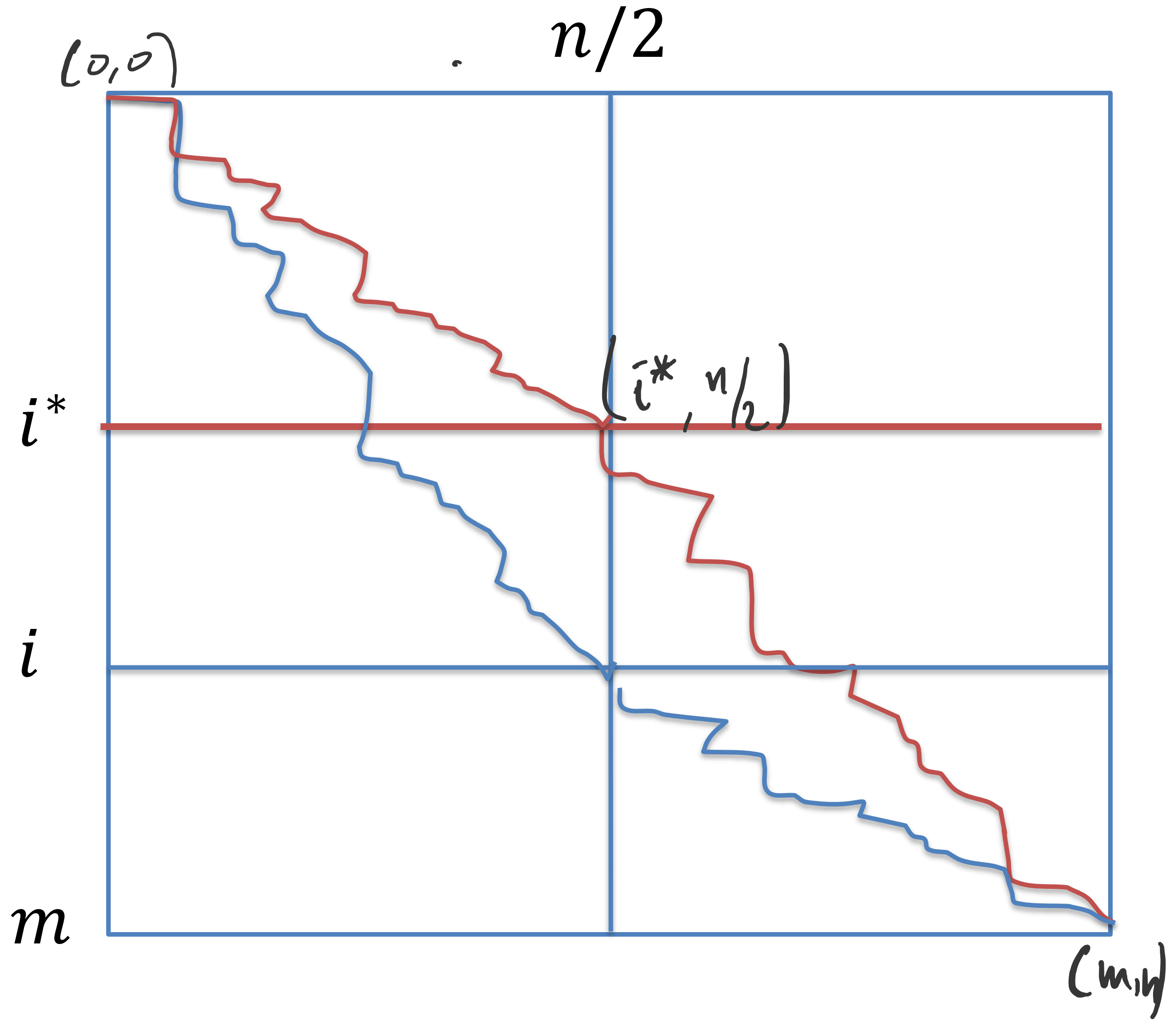


Space Efficient Alignment – Second Attempt

Alignment is a path from source $(0,0)$ to target (m,n) in edit graph

Maximum weight path from $(0{,}0)$ to (m,n) passes through $(i^*, n/2)$

Question: What is i^*?



Space Efficient Alignment – Second Attempt

$\boxed{wt(i)}$ is the weight of max weight path from $(0,0)$ ~~to~~ to (m,n) passing through $(i, n/2)$.

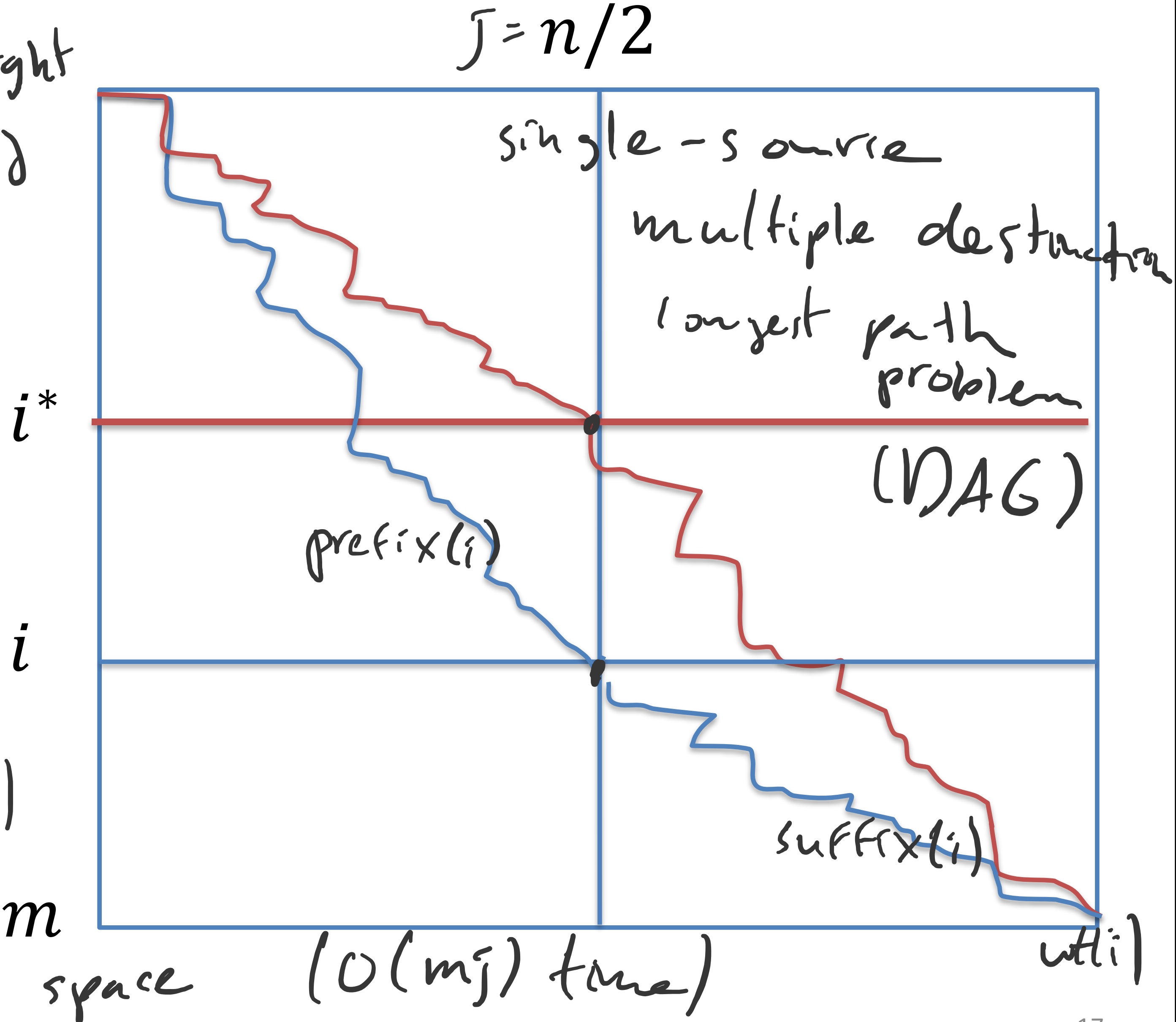
$$i^* = \arg\max_{0 \le i \le m} wt(i)$$

How to compute $wt(i)$?

$wt(i) = prefix(i) + suffix(i)$

$prefix(i)$: max weight of path from $(0,0)$ to $(i, n/2)$

$prefix(0), \ldots, prefix(m)$ ↖ $O(m)$ space



$(O(mj)$ time$)$

Space Efficient Alignment – Second Attempt

How to compute suffix(i)

Want: O(m) space

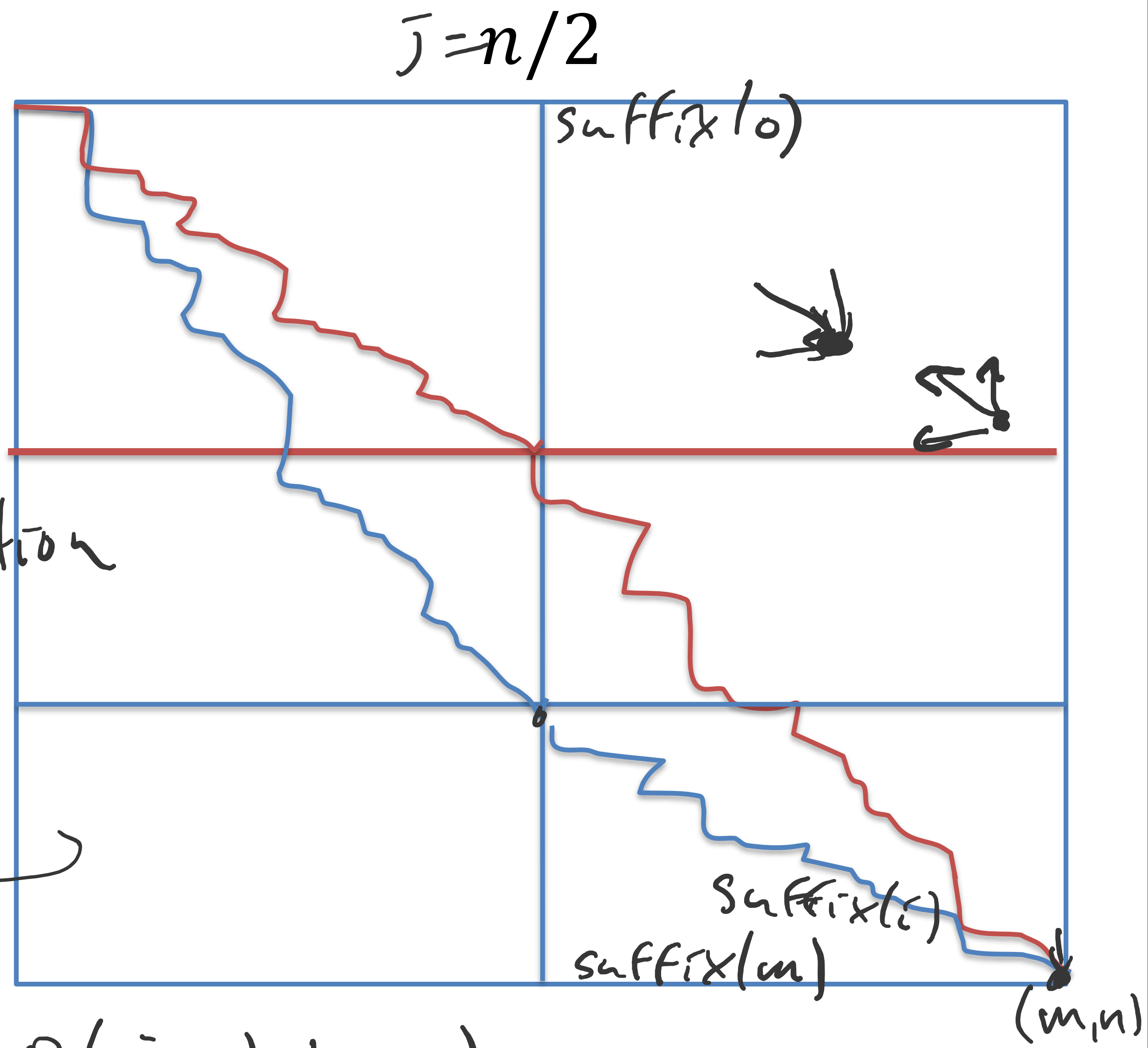
O(jm) time

Key idea: "go backwards" by reversing direction of edges

$\underbrace{\text{suffix}(0), \ldots, \text{suffix}(m)}$

O(m) space

O(n/2 m) time (O(jm) time)



Linear-Space Sequence Alignment

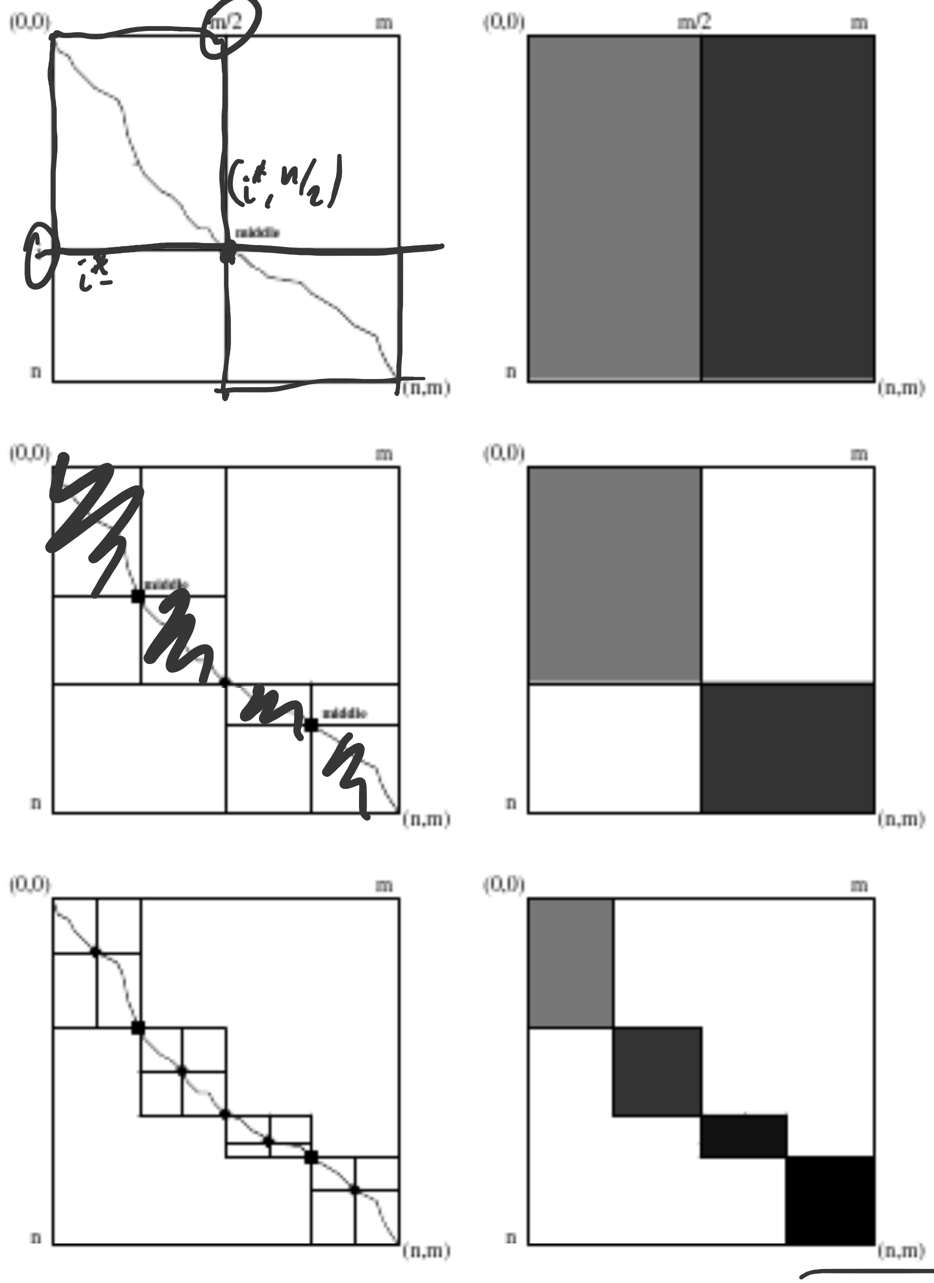


Figure 7.3 Space-efficient sequence alignment. The computational time (i.e., the area of the solid rectangles) decreases by a factor of 2 at every iteration.

```
Hirschberg$(i, j, i', j')$
1.  if $j' - j > 1$
2.      $i^* \leftarrow \arg\max_{i \le i'' \le i'} \mathrm{wt}(i'')$
3.      Report $(i^*, j + \frac{j'-j}{2})$
4.      Hirschberg$(i, j, i^*, j + \frac{j'-j}{2})$
5.      Hirschberg$(i^*, j + \frac{j'-j}{2}, i', j')$
```

(i,j) top left vertex

(i',j') bottom right vertex

middle column

i : row index

i	$\mathrm{wt}(i)$ =	prefix(i) +	suffix(i)
0			
1			
⋮			
m			

$O(m)$ space

$O(m)$ space

$O(m)$ space

Linear-Space Sequence Alignment

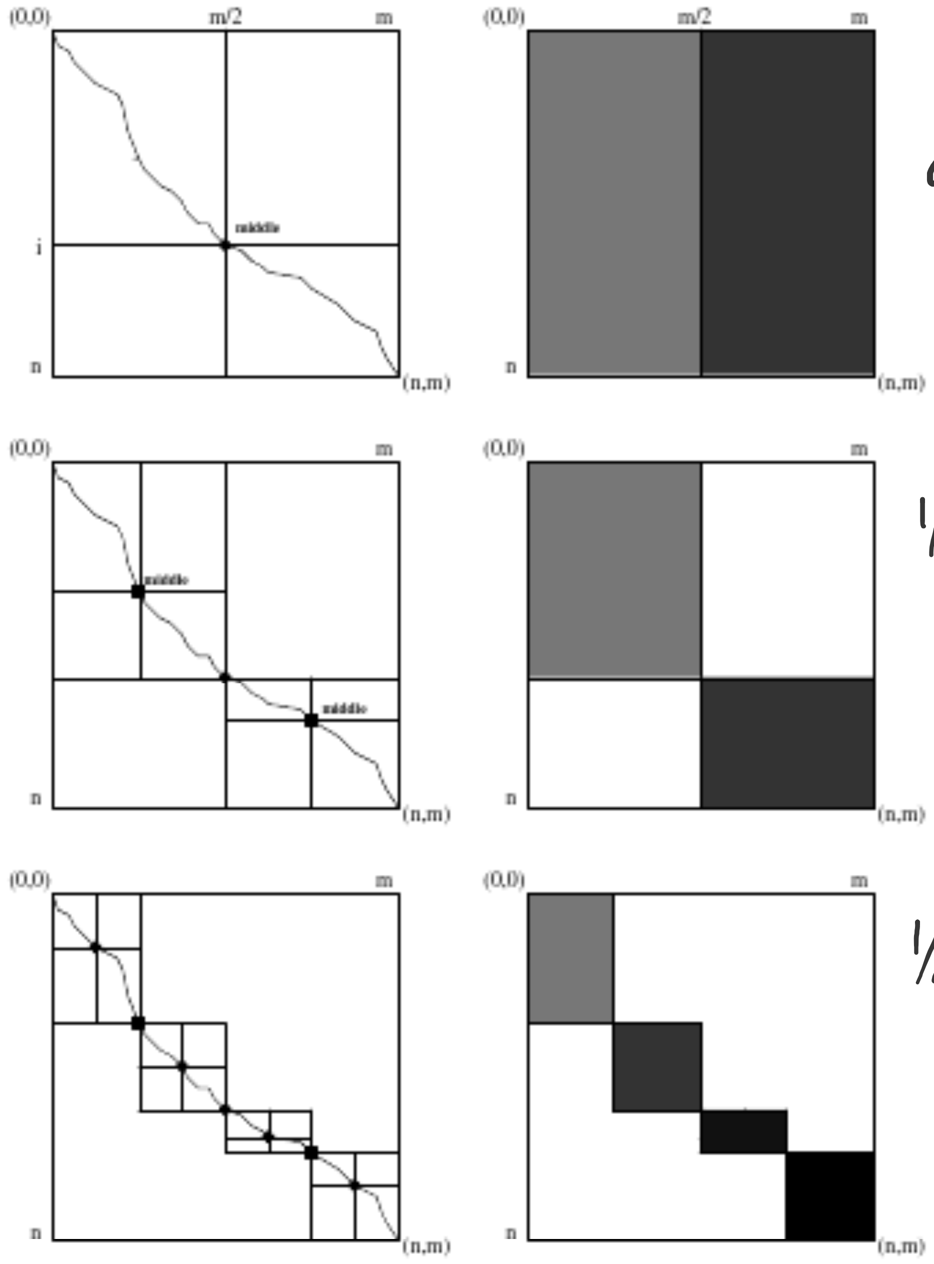


Figure 7.3 Space-efficient sequence alignment. The computational time (i.e., the area of the solid rectangles) decreases by a factor of 2 at every iteration.

area
O(mn)

1/2 area

1/4 area

```
Hirschberg$(i, j, i', j')$
1. if $j' - j > 1$
2.     $i^* \leftarrow \arg\max_{i \le i'' \le i'} \mathrm{wt}(i'')$
3.     Report $(i^*, j + \frac{j'-j}{2})$
4.     Hirschberg$(i, j, i^*, j + \frac{j'-j}{2})$
5.     Hirschberg$(i^*, j + \frac{j'-j}{2}, i', j')$
```

Time:

area + area/2 + area/4 + …

= area $\underbrace{(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots)}$ geometric series

≤ 2 × $\underline{\text{area}}$ = O(mn) quadratic

Space: O(m)

Linear-Space Sequence Alignment

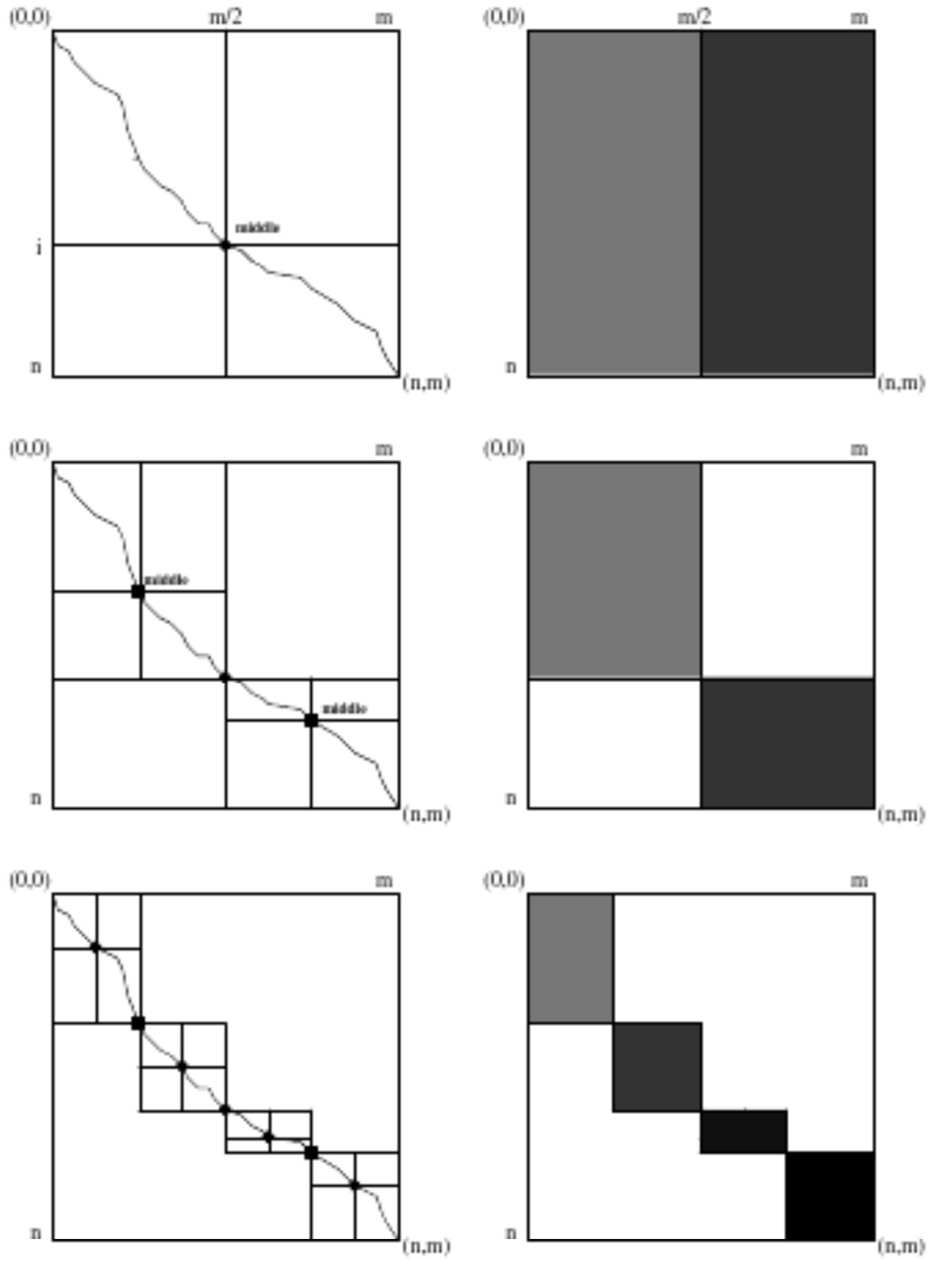


Figure 7.3 Space-efficient sequence alignment. The computational time (i.e., the area of the solid rectangles) decreases by a factor of 2 at every iteration.

```
Hirschberg($i, j, i', j'$)
1.  if $j' - j > 1$
2.      $i^* \leftarrow \arg\max_{i \le i'' \le i'} \mathrm{wt}(i'')$
3.      Report $(i^*, j + \frac{j'-j}{2})$
4.      Hirschberg($i, j, i^*, j + \frac{j'-j}{2}$)
5.      Hirschberg($i^*, j + \frac{j'-j}{2}, i', j'$)
```

Time:

area + area/2 + area/4 + …

= area (1 + ½ + ¼ + ⅛ + …)

≤ 2 × area = O(mn)

Space: O(m)

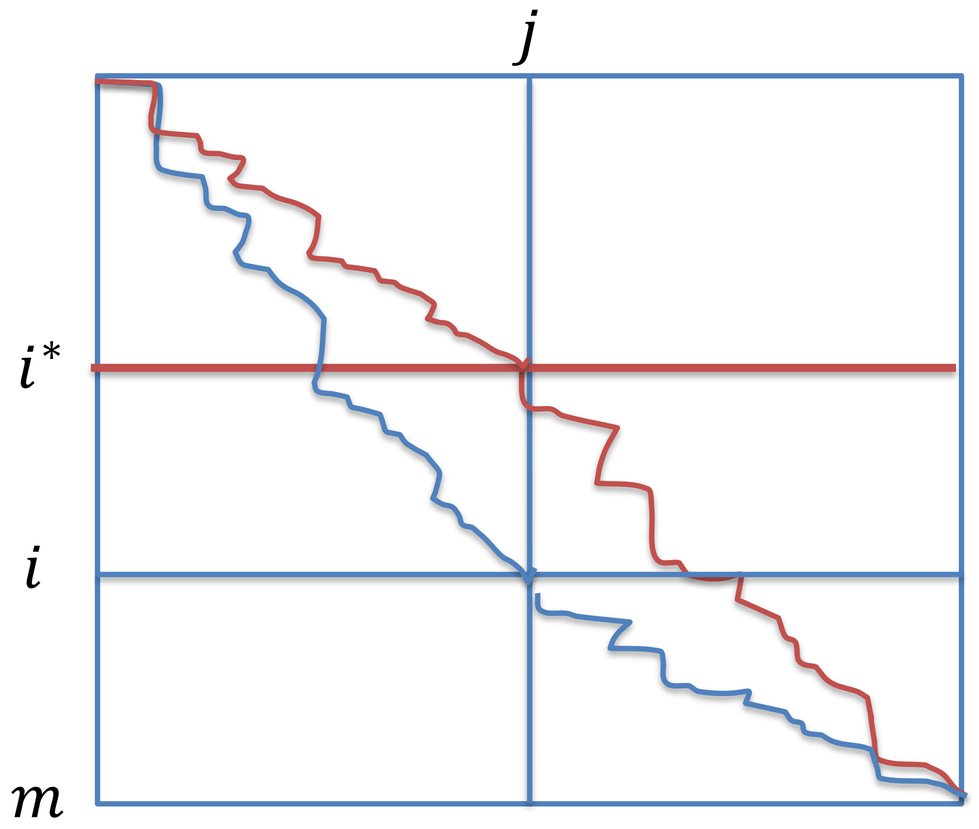
Question: How to reconstruct alignment from reported vertices?

Hirschberg Algorithm: Reversing Edges Necessary?

Max weight path from $(0,0)$ to (m,n) through $(i^*, n/2)$

$$i^* = \underset{0\le i\le m}{\arg\max}\{\, \text{prefix}(i) + \text{suffix}(i)\,\}$$

Compute $\{\text{prefix}(i) \mid 0 \le i \le m\}$ in $O(mj)$ time and $O(m)$ space, by starting from $(0,0)$ to (m,j) keeping only two columns in memory. [**single-source multiple destinations**]



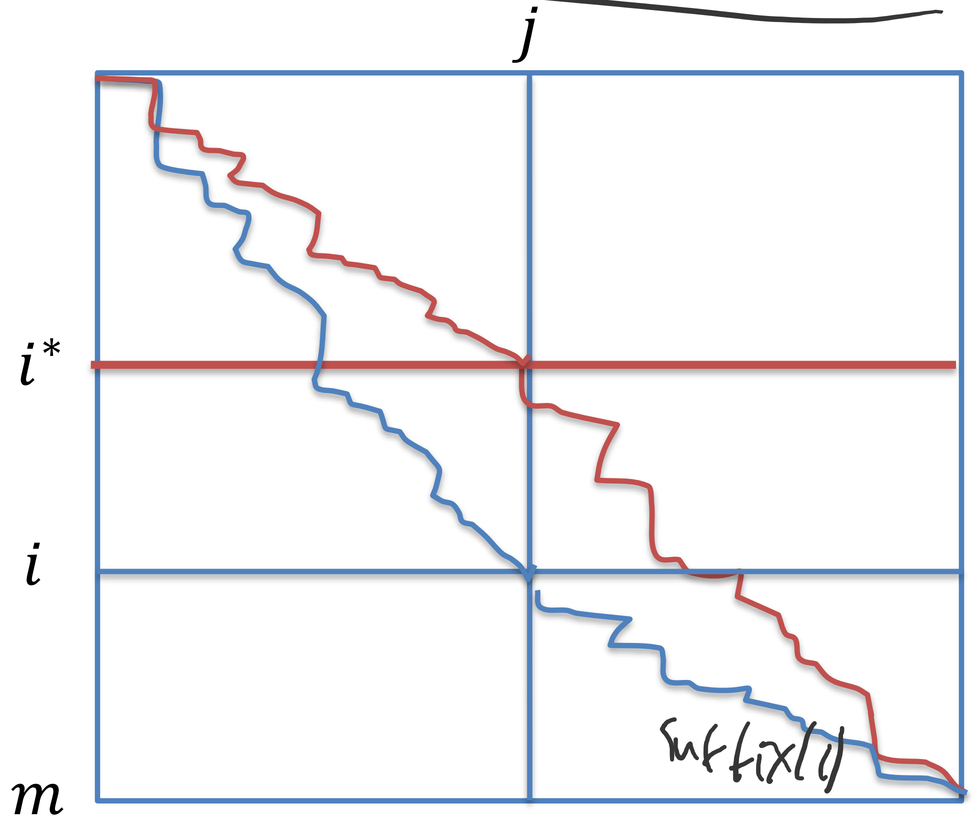
Hirschberg Algorithm: Reversing Edges Necessary?

Max weight path from $(0,0)$ to (m, n) through $(i^*, n/2)$

$$i^* = \arg\max_{0 \le i \le m} \{\, \text{prefix}(i) + \text{suffix}(i) \,\}$$

Compute $\{\text{prefix}(i) \mid 0 \le i \le m\}$ in $O(mj)$ time and $O(m)$ space, by starting from $(0,0)$ to (m, j) keeping only two columns in memory. [**single-source multiple destinations**]

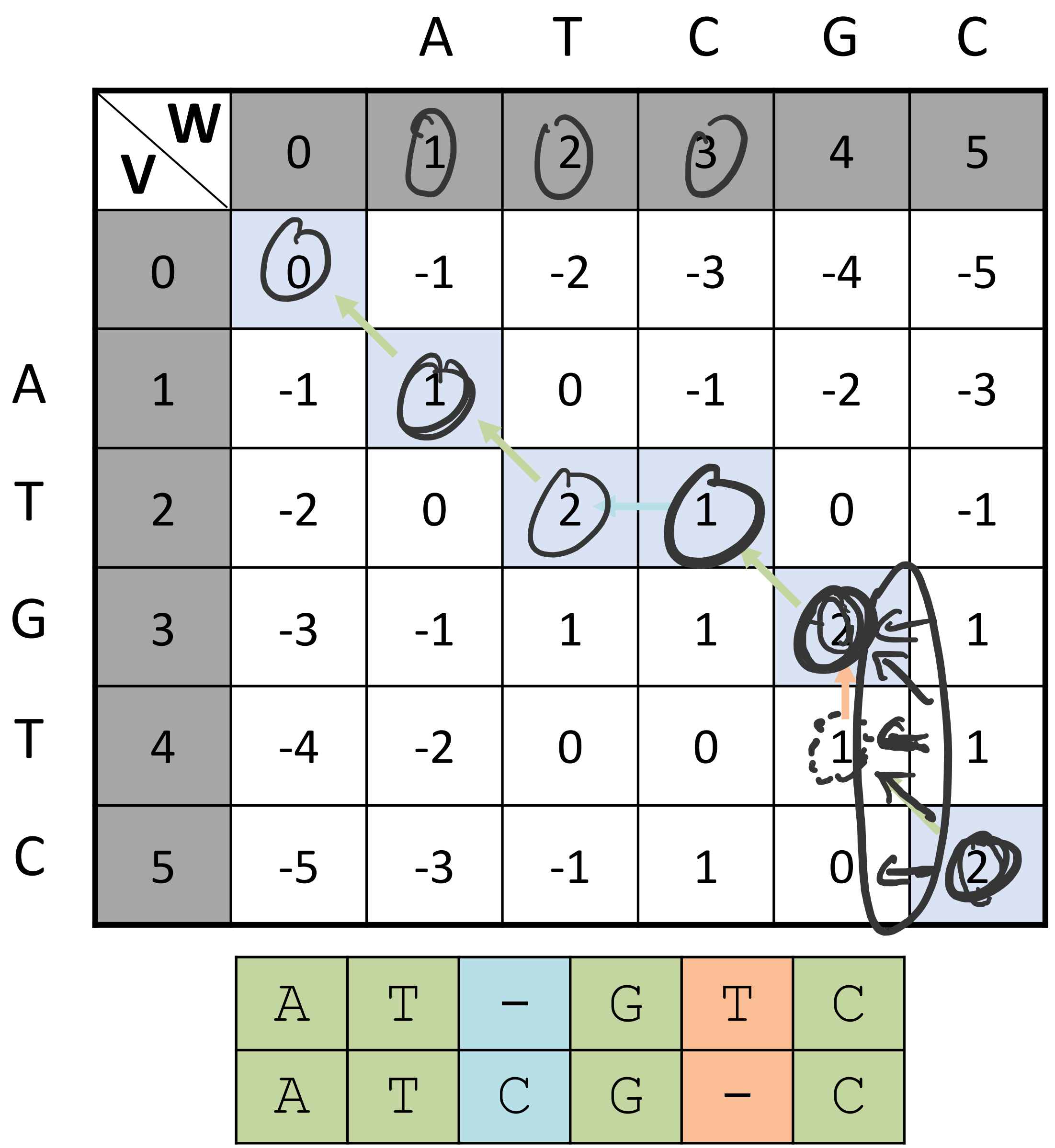
Want: Compute $\{\text{suffix}(i) \mid 0 \le i \le m\}$ in $O(mj)$ time and $O(m)$ space



Doing a longest path from each (i, j) to (m, n) (for all $0 \le i \le m$) will not achieve desired running time!

Reversing edges enables single-source multiple destination computation in desired time and space bound!

Hirschberg Algorithm: Reconstructing Alignment



			A	T	C	G	C
	V \ W	0	1	2	3	4	5
	0	0	-1	-2	-3	-4	-5
A	1	-1	1	0	-1	-2	-3
T	2	-2	0	2	1	0	-1
G	3	-3	-1	1	1	2	1
T	4	-4	-2	0	0	1	1
C	5	-5	-3	-1	1	0	2

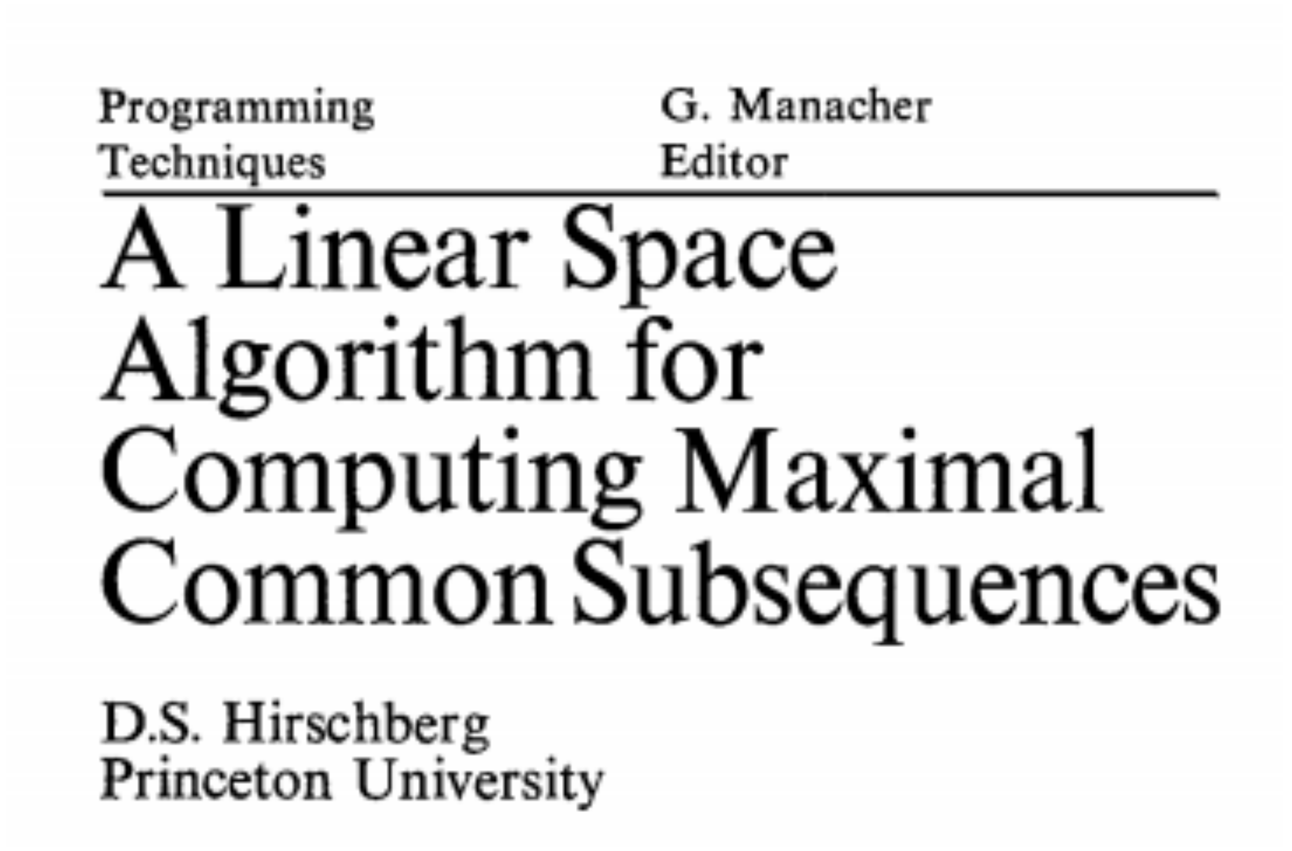
A	T	-	G	T	C
A	T	C	G	-	C

```
Hirschberg($i, j, i', j'$)
1.  if $j' - j > 1$
2.      $i^* \leftarrow \underset{0 \le i \le m}{\arg\max}\ \mathrm{wt}(i)$
3.      Report $(i^*, j + \frac{j'-j}{2})$
4.      Hirschberg($i, j, i^*, j + \frac{j'-j}{2}$)
5.      Hirschberg($i^*, j + \frac{j'-j}{2}, i', j'$)
```

Problem: Given reported vertices and scores $\{(i_0, 0, s_0), \dots, (i_n, n, s_n)\}$, find intermediary vertices.

Transposing matrix does not help, because gaps could occur in both input sequences (and there might be multiple opt. alignments)

Linear Space Alignment – The Hirschberg Algorithm



Programming Techniques

G. Manacher Editor

A Linear Space Algorithm for Computing Maximal Common Subsequences

D.S. Hirschberg
Princeton University

Dan Hirschberg

Professor of Computer Science & EECS
UC Irvine Senate Parliamentarian

Outline

1. Recap of global, fitting, local and gapped alignment
2. Space-efficient alignment
3. Subquadratic time alignment

Reading:

- Jones and Pevzner. Chapters 7.1-7.4
- Lecture notes

global, local, fitting

$\bar{v} \in \Sigma^{m}$

$\bar{w} \in \Sigma^{n}$

$O(mn)$ time

$n > m \Rightarrow O(n^2)$

[illegible]

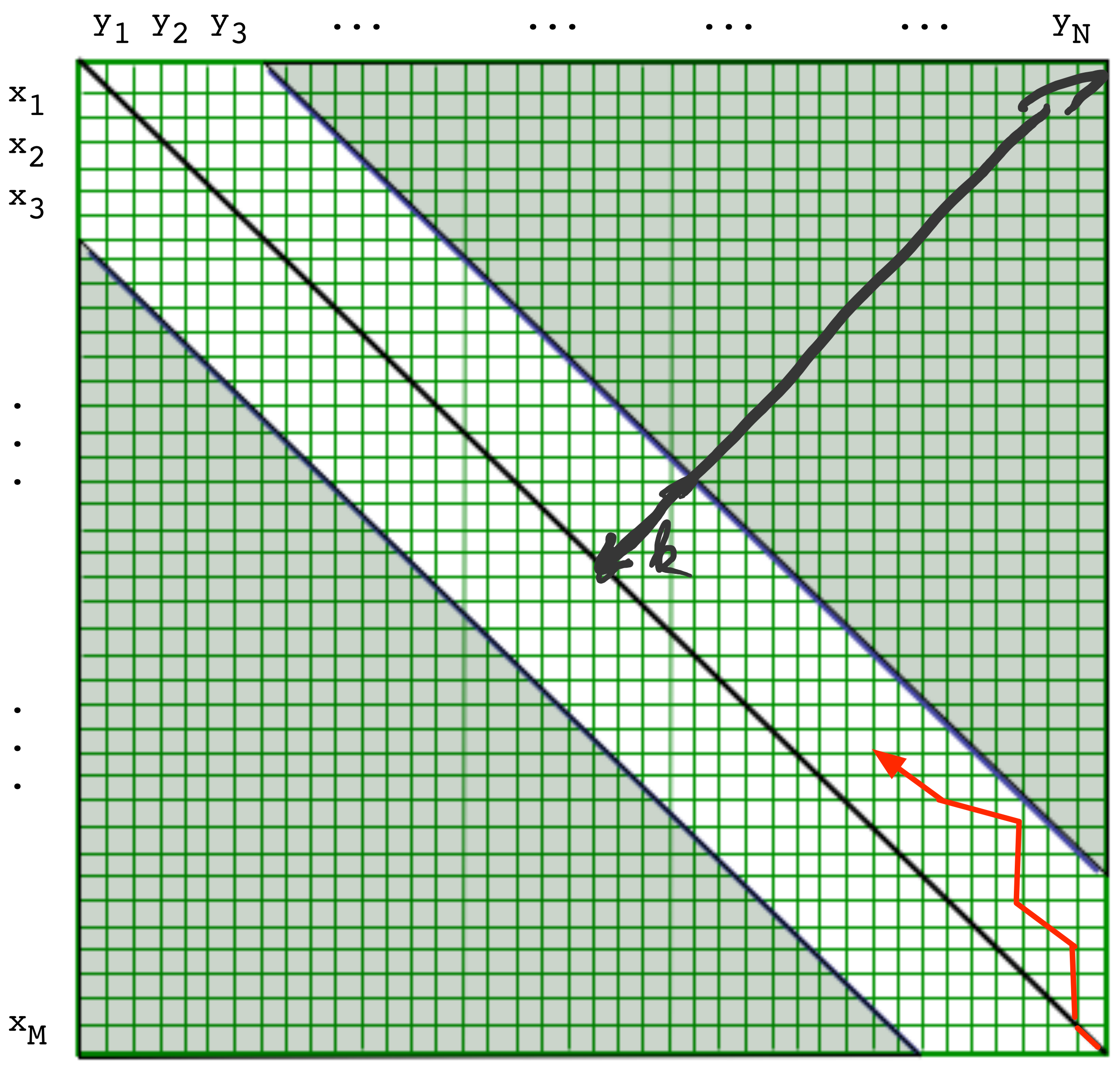
Banded Alignment

Constraint path to band of width k around diagonal

Running time: $O(nk)$

Gives a good approximation of highly identical sequences

$\tilde{v}$ $\bar{w}$ similar



Constrain traceback to band of DP matrix (penalize big gaps)

Figure source: http://jinome.stanford.edu/stat366/pdfs/stat366_win0607_lecture04.pdf

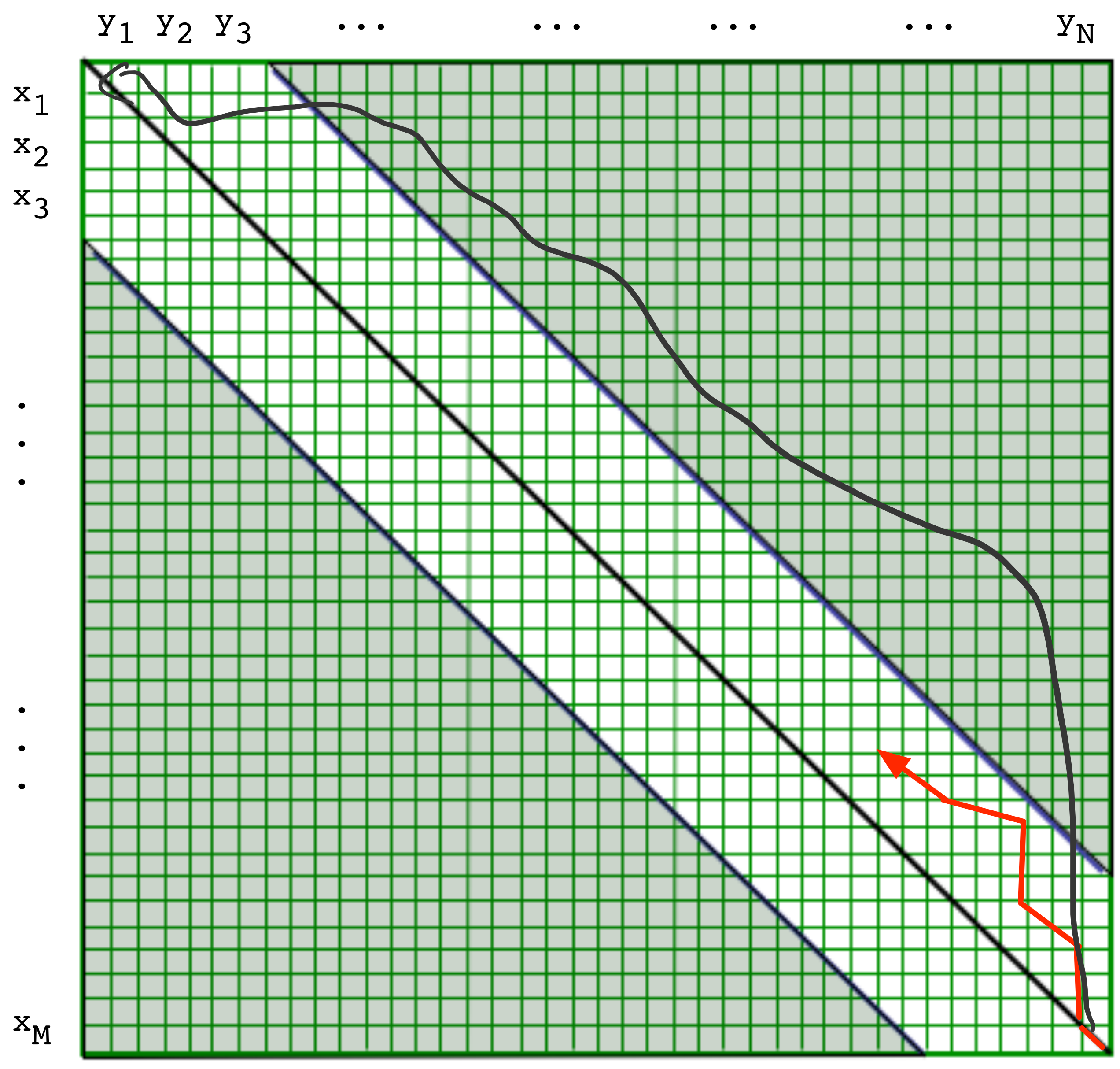
Banded Alignment

Constraint path to band of width k around diagonal

Running time: $O(nk)$

Gives a good approximation of highly identical sequences

Question: How to change recurrence to accomplish this?

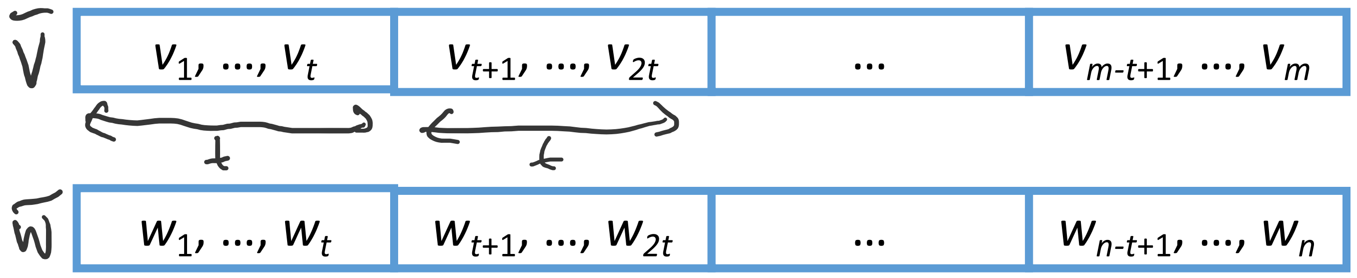


Constrain traceback to band of DP matrix (penalize big gaps)

Figure source: http://jinome.stanford.edu/stat366/pdfs/stat366_win0607_lecture04.pdf

Block Alignment

Divide input sequences into blocks of length t



Block Alignment

Divide input sequences into blocks of length t

$v_1, \ldots, v_t$	$v_{t+1}, \ldots, v_{2t}$	…	$v_{m-t+1}, \ldots, v_m$

$w_1, \ldots, w_t$	$w_{t+1}, \ldots, w_{2t}$	…	$w_{n-t+1}, \ldots, w_n$

Require that paths in edit graph pass through **corners** of blocks

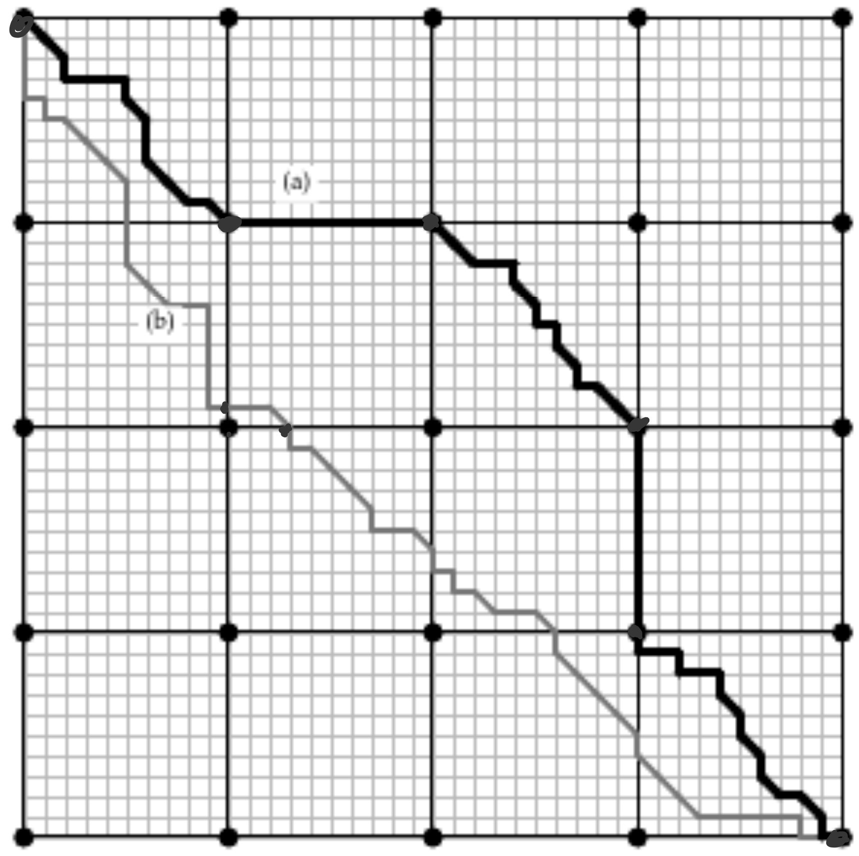


Figure 7.4 Two paths in a 40×40 grid partitioned into 16 subgrids of size 10×10. The black path (a) is a block path, while the gray path (b) is not.

Block Alignment

Divide input sequences into blocks of length t

$v_1, \ldots, v_t$	$v_{t+1}, \ldots, v_{2t}$	…	$v_{m-t+1}, \ldots, v_m$

$w_1, \ldots, w_t$	$w_{t+1}, \ldots, w_{2t}$	…	$w_{n-t+1}, \ldots, w_n$

Require that paths in edit graph pass through **corners** of blocks

$$\boxed{s[i,j]} = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] - \sigma, & \text{if } i > 0, \\ s[i,j-1] - \sigma, & \text{if } j > 0, \\ s[i-1,j-1] + \boxed{\beta(i,j)}, & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

$0 \le i,j \le t$ and $\beta(i,j)$ is max score alignment between block i of **v** and block j of **w**

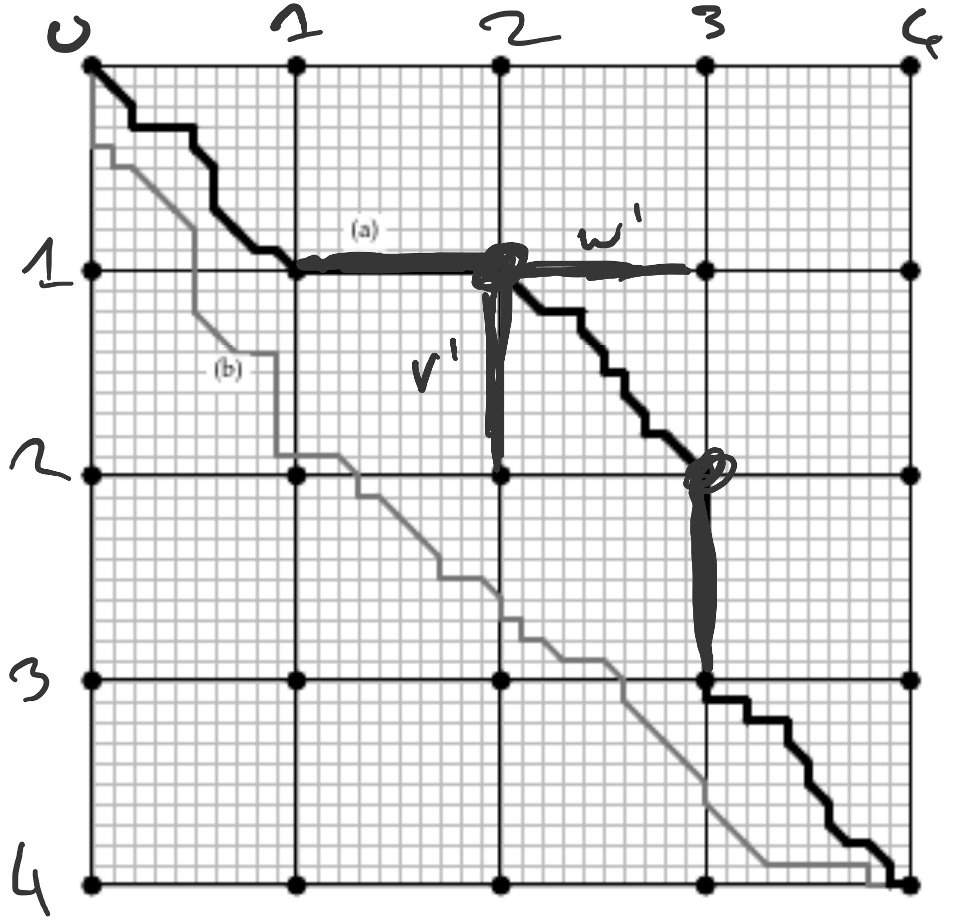


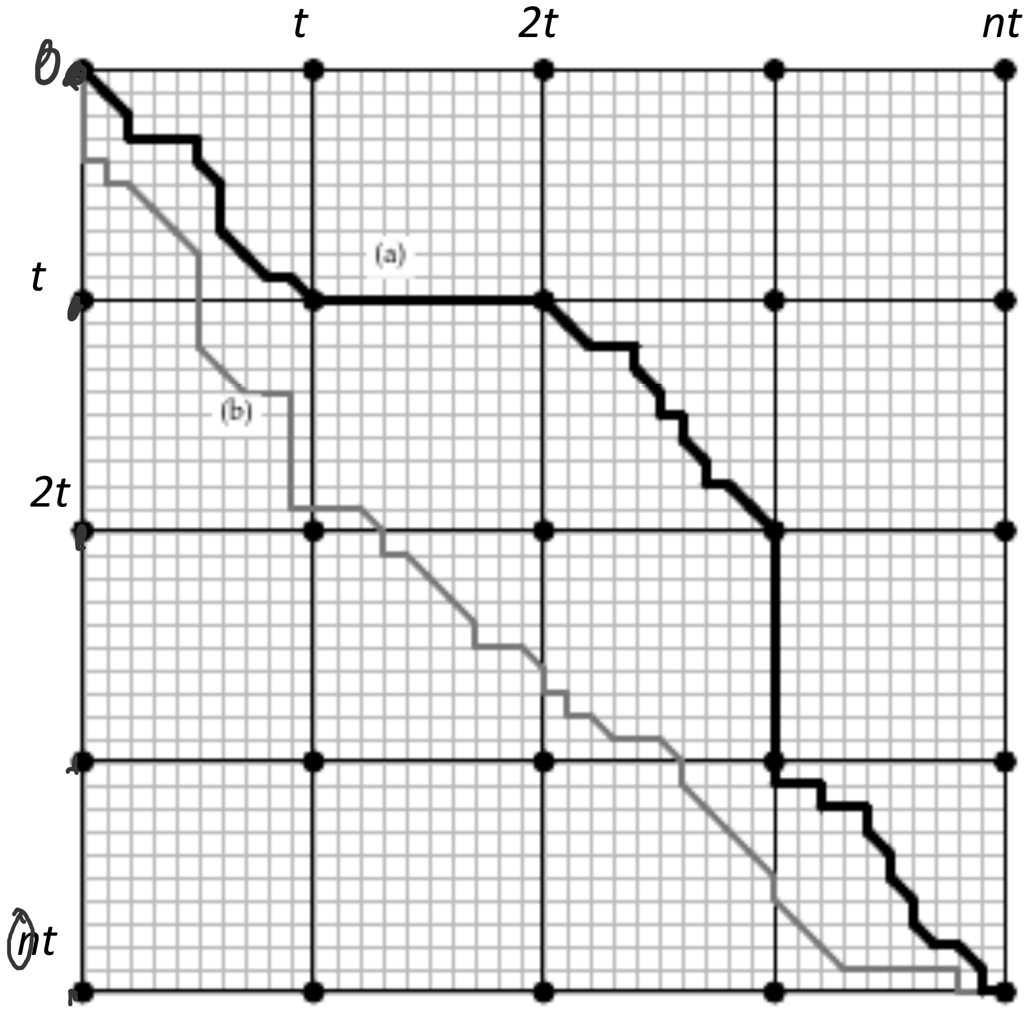
Figure 7.4 Two paths in a 40 × 40 grid partitioned into 16 subgrids of size 10 × 10. The black path (a) is a block path, while the gray path (b) is not.

Block Alignment – First Attempt: Pre-compute $\beta(i,j)$

$0 \leq i,j \leq n/t$ and $\beta(i,j)$ is max score alignment between block i of $\mathbf{v}$ and block j of $\mathbf{w}$

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] - \sigma, & \text{if } i > 0, \\ s[i,j-1] - \sigma, & \text{if } j > 0, \\ s[i-1,j-1] + \beta(i,j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

i



Block Alignment – First Attempt: Pre-compute $\beta(i,j)$

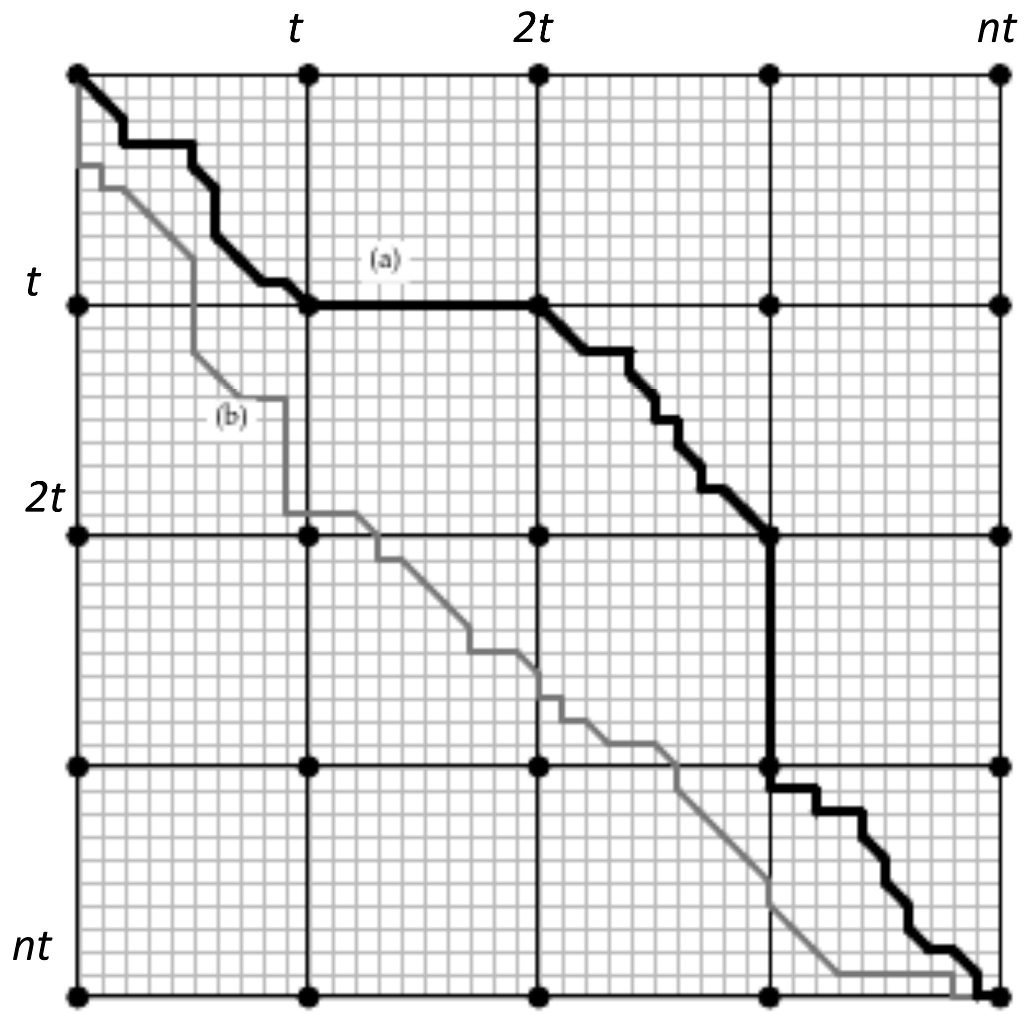
$0 \le i,j \le n/t$ and $\beta(i,j)$ is max score alignment between block i of $\mathbf{v}$ and block j of $\mathbf{w}$

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] - \sigma, & \text{if } i > 0, \\ s[i,j-1] - \sigma, & \text{if } j > 0, \\ s[i-1,j-1] + \beta(i,j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

Question:
How much time to compute all $\beta(i,j)$?

$|\bar{v}'| = |\bar{w}'| = t$

$O(t^2)$



Block Alignment – First Attempt: Pre-compute $\beta(i,j)$

$0 \le i,j \le n/t$ and $\beta(i,j)$ is max score alignment between block i of $\mathbf{v}$ and block j of $\mathbf{w}$

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] - \sigma, & \text{if } i > 0, \\ s[i,j-1] - \sigma, & \text{if } j > 0, \\ s[i-1,j-1] + \beta(i,j), & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

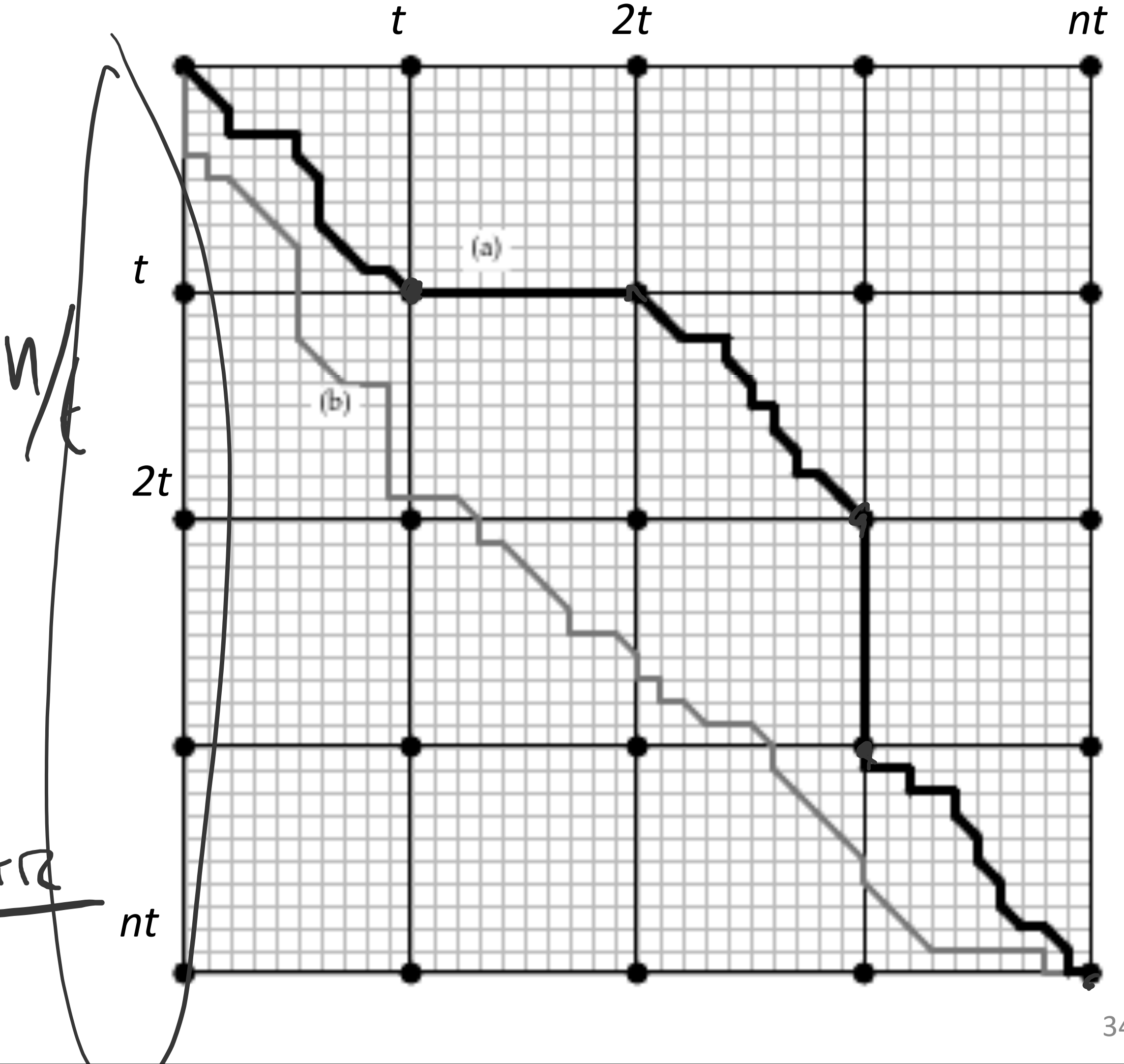
Question:
How much time to compute all $\beta(i,j)$?

Computing $\beta(i,j)$ takes $O(t^2)$ time

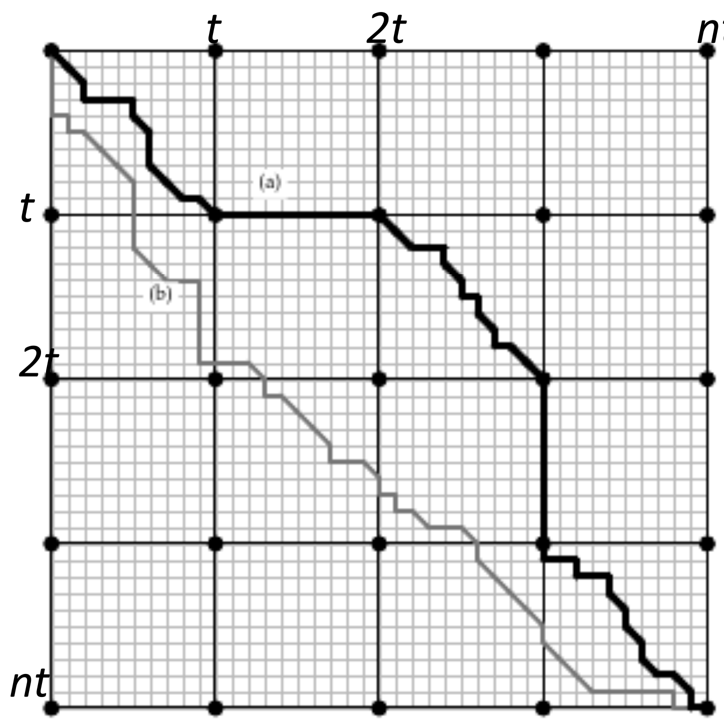
There are $n/t \times n/t$ values $\beta(i,j)$

Total: $O\left(\frac{n}{\cancel{t}} \times \frac{n}{\cancel{t}} \times \cancel{t^2}\right) = O(n^2)$ time

quadratic



Block Alignment – Four Russians Technique



~~Pre-compute and store all β_{ij}~~

$O(n^2)$ time

Pre-compute and store ***all*** max weight alignments $S[\mathbf{v}', \mathbf{w}']$ of ***all*** pairs $(\mathbf{v}', \mathbf{w}')$ of length t strings

① max all strings of length t

$(\bar{v}', \bar{w}') \in \Sigma^t \times \Sigma^t$

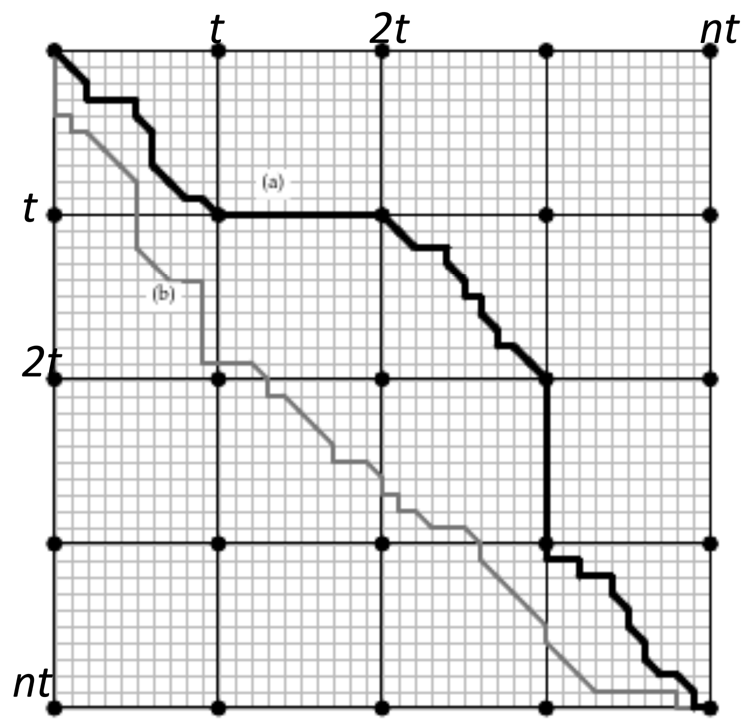
② Make all pairs

Algorithm:

1. Precompute $S[\mathbf{v}', \mathbf{w}']$ where $\mathbf{v}', \mathbf{w}' \in \Sigma^t$
2. Compute block alignment between $\mathbf{v}$ and $\mathbf{w}$ using S

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] - \sigma, & \text{if } i > 0, \\ s[i,j-1] - \sigma, & \text{if } j > 0, \\ s[i-1,j-1] + S[v(i), w(j)], & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

Block Alignment – Four Russians Technique



~~Pre-compute and store all β$_{ij}$~~

Pre-compute and store ***all*** max weight alignments $S[\mathbf{v}', \mathbf{w}']$ of ***all*** pairs $(\mathbf{v}', \mathbf{w}')$ of length *t* strings

Question: How to choose *t* for DNA?

$|\Sigma| = 4$

Algorithm:

1. Precompute $S[\mathbf{v}', \mathbf{w}']$ where $\mathbf{v}', \mathbf{w}' \in \Sigma^t$
2. Compute block alignment between $\mathbf{v}$ and $\mathbf{w}$ using S

$$s[i,j] = \max \begin{cases} 0, & \text{if } i = 0 \text{ and } j = 0, \\ s[i-1,j] - \sigma, & \text{if } i > 0, \\ s[i,j-1] - \sigma, & \text{if } j > 0, \\ s[i-1,j-1] + S[v(i), w(j)], & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

Block Alignment – Four Russians Technique

Question: How to choose *t* for DNA? $\Sigma = \{A, T, C, G\}$

$$\boxed{t = \frac{\log_2 n}{4}}$$

$n =$ ~~32~~, ~~128~~ 256

$$t = \frac{\log_2 256}{4} = \frac{8}{4} = 2$$

① make all strings of length t. (from Σ)

$$4^t = 4^{\log_2(n)/4}$$
$$= (2^2)^{\log_2(n)/4}$$
$$= 2^{\log_2(n)/2}$$
$$= (2^{\log_2 n})^{1/2} = n^{1/2} = \boxed{\sqrt{n}}$$

② number of pairs of strings of length t.

$$\sqrt{n} \cdot \sqrt{n} = n$$

③ compute alignment for each pair $O(t^2)$

$$\boxed{①\cdot②\cdot③: \overset{O(nt^2)}{=} O(n (\log_2 n)^2) = O(n \log^2 n)}$$

④ Use look-up table S' to compute block alignment

$$S'[\underbrace{\bar{v}'}_{t}, \underbrace{\bar{w}'}_{t}] \qquad t = \frac{\log_2 n}{4} = O(\log n)$$

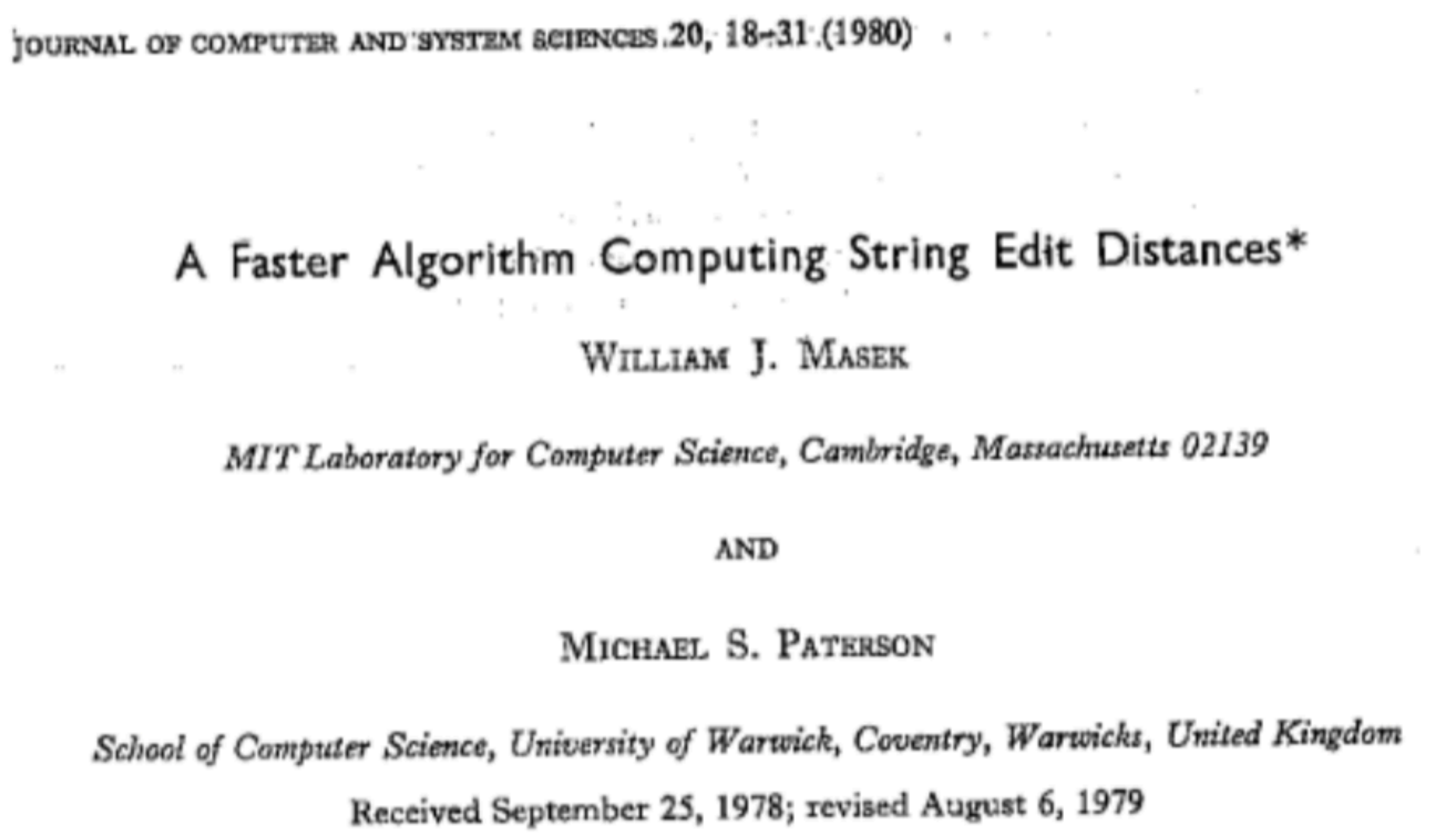
$$\boxed{\begin{array}{c}\text{indexing } O(\log n)\\ \text{key has length } 2t\end{array}}$$

each look up in S takes $O(\log n)$ time

⑤ Number of look ups $\frac{n}{t} \times \frac{n}{t}$

$$O\left(\frac{n}{t} \cdot \frac{n}{t} \cdot \log n\right) = O\left(\frac{n}{\log n} \cdot \frac{n}{\cancel{\log n}} \cdot \cancel{\log n}\right)$$
$$= O(n^2/\log n)$$

Fastest Subquadratic Alignment* Algorithm



JOURNAL OF COMPUTER AND SYSTEM SCIENCES 20, 18-31 (1980)

A Faster Algorithm Computing String Edit Distances*

WILLIAM J. MASEK

MIT Laboratory for Computer Science, Cambridge, Massachusetts 02139

AND

MICHAEL S. PATERSON

School of Computer Science, University of Warwick, Coventry, Warwicks, United Kingdom

Received September 25, 1978; revised August 6, 1979

① weakly subquadratic

Edit distance in $O(n^2/\log n)$ time

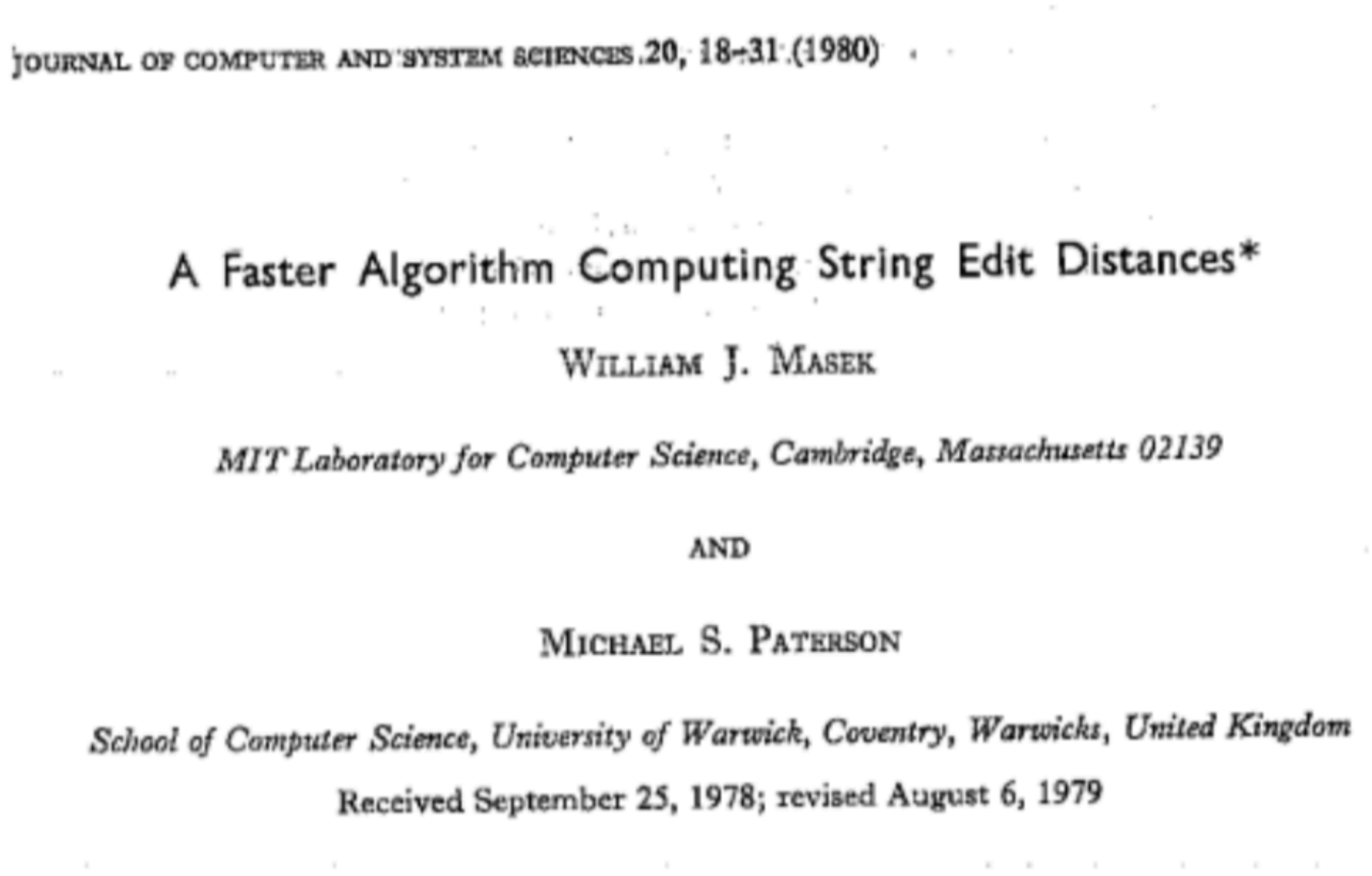
Barely subquadratic!

Want: $O(n^{2-\varepsilon})$ time where $\varepsilon > 0$

$n^2/\log n$ is slower than $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$

*for edit distance

Fastest Subquadratic Alignment* Algorithm



JOURNAL OF COMPUTER AND SYSTEM SCIENCES 20, 18–31 (1980)

A Faster Algorithm Computing String Edit Distances*

WILLIAM J. MASEK

MIT Laboratory for Computer Science, Cambridge, Massachusetts 02139

AND

MICHAEL S. PATERSON

School of Computer Science, University of Warwick, Coventry, Warwicks, United Kingdom

Received September 25, 1978; revised August 6, 1979

Edit distance in $O(n^2/\log n)$ time

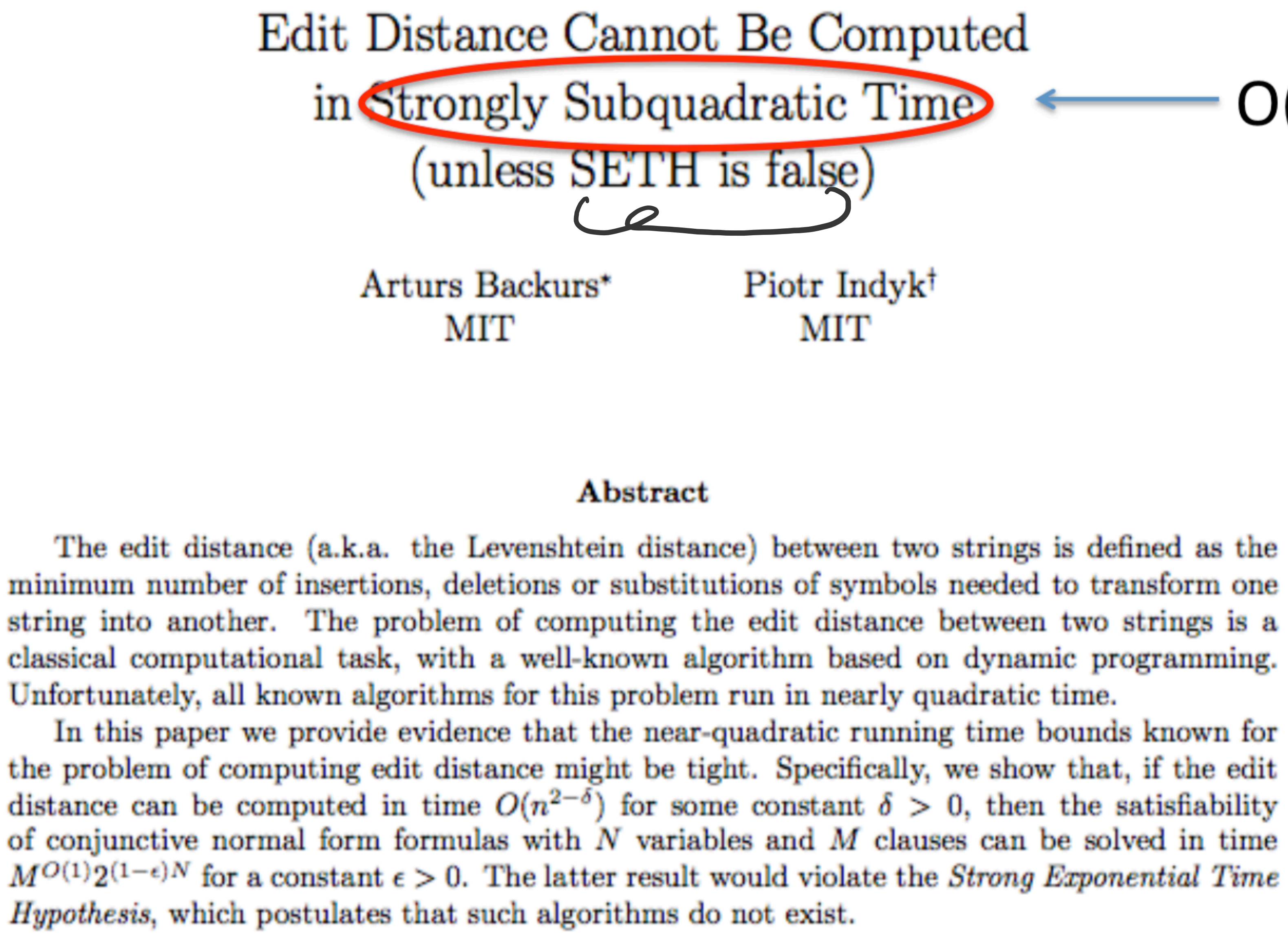
Barely subquadratic!

Want: $O(n^{2-\varepsilon})$ time where $\varepsilon > 0$

Question: Is $n^{2-\varepsilon}$ in $O(n^2/\log n)$ for any $\varepsilon > 0$? lecture notes

*for edit distance

Hardness Result for Edit Distance [STOC 2015]



Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)

Arturs Backurs[*] MIT

Piotr Indyk[†] MIT

Abstract

The edit distance (a.k.a. the Levenshtein distance) between two strings is defined as the minimum number of insertions, deletions or substitutions of symbols needed to transform one string into another. The problem of computing the edit distance between two strings is a classical computational task, with a well-known algorithm based on dynamic programming. Unfortunately, all known algorithms for this problem run in nearly quadratic time.

In this paper we provide evidence that the near-quadratic running time bounds known for the problem of computing edit distance might be tight. Specifically, we show that, if the edit distance can be computed in time $O(n^{2-\delta})$ for some constant $\delta > 0$, then the satisfiability of conjunctive normal form formulas with N variables and M clauses can be solved in time $M^{O(1)}2^{(1-\epsilon)N}$ for a constant $\epsilon > 0$. The latter result would violate the *Strong Exponential Time Hypothesis*, which postulates that such algorithms do not exist.

$\longleftarrow$ $O(n^{2\text{-}\varepsilon})$ time where $\varepsilon > 0$

BRAINIAC

For 40 years, computer scientists looked for a solution that doesn't exist [1]

SHUTTERSTOCK

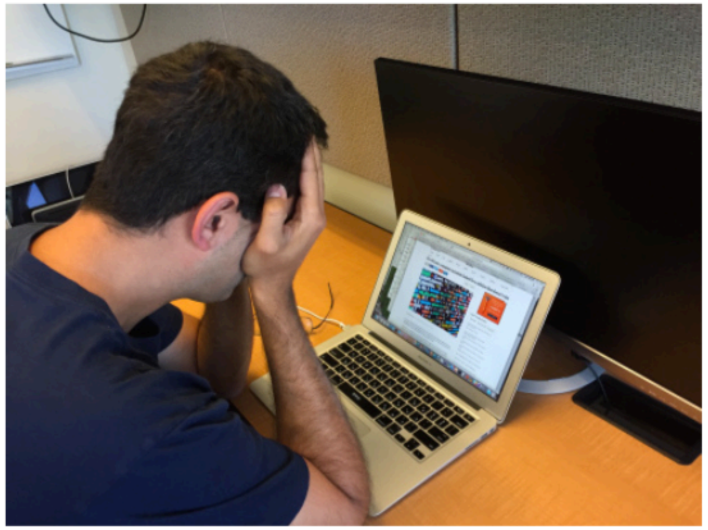
By Kevin Hartnett | GLOBE CORRESPONDENT AUGUST 10, 2015

For 40 years, computer scientists have tried in vain to find a faster way to do an important calculation known as "edit distance." Thanks to groundbreaking work from two researchers at MIT, they now know the reason they've continually failed is because a faster method is actually impossible to create.

In biology n does not go to infinity [2]

August 14, 2015 in reviews | Tags: complexity theory, edit distance, Needleman-Wunsch algorithm, strong exponential time hypothesis

I recently read a "*brainiac*" column in the Boston Globe titled "For 40 years, computer scientists looked for a solution that doesn't exist" that caused me to facepalm so violently I now have pain in my right knee.



[1] Boston Globe, Aug 10, 2015
[2] Bits of DNA Blog, Lior Pachter

Take Home Messages

1. Global alignment in $O(mn)$ time and $O(m)$ space
   - Hirschberg algorithm
2. Block alignment can be done in subquadratic time
   - Four Russians Technique: $O(n^2/\log n)$ time
3. Global alignment cannot be done in $O(n^{2-\varepsilon})$ time under SETH

Reading:

- Jones and Pevzner. Chapters 7.1-7.4
- Lecture notes