Computing Alignment Plots Efficiently

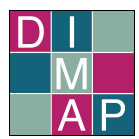
. . . in theory and practice

Peter Krusche Alexander Tiskin

Department of Computer Science
University of Warwick, Coventry, CV4 7AL, UK

LSD 2010

THE UNIVERSITY OF WARWICK



Motivation

This talk is about *loss-free* local alignment of (biological) sequences.

Our loss-free alignment algorithms find *all* local alignments of two input sequences.

This is computationally very demanding.

What is in this talk?

A computational technique for computing multiple local alignments at the same time.

Some discussion about its efficient implementation.

Speedup results from different types of parallelism.

Outline

String Terminology

A *string* is a sequence of characters from an alphabet Σ.

Example (Genome Data)

Strings are sequences of characters from { A, C, G, T }

CAGAGGATGAGGATG

String Terminology

Contiguous subsequences are called *substrings/windows/factors*.

CAGAGGATGAGGATG

We also consider not necessarily contiguous *subsequences*.

CAGAGGATGAGGATG

String Terminology

Contiguous subsequences are called *substrings/windows/factors*.

CAGAGGATGAGGATG

We also consider not necessarily contiguous *subsequences*.

CAGAGGATGAGGATG

String Comparison with Errors

Hamming distance: count mismatches.

dist(bbbababababba, abbbbabaaaba) = 3

Used e.g. in dot-plots for local comparison.

String Alignment

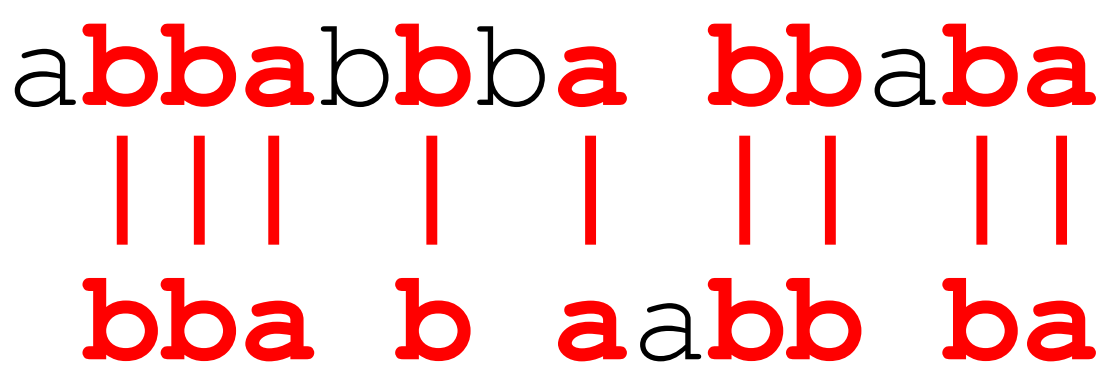
Align the maximum number of letters, preserving order:

abbabbbabbaba

bbabaabbba

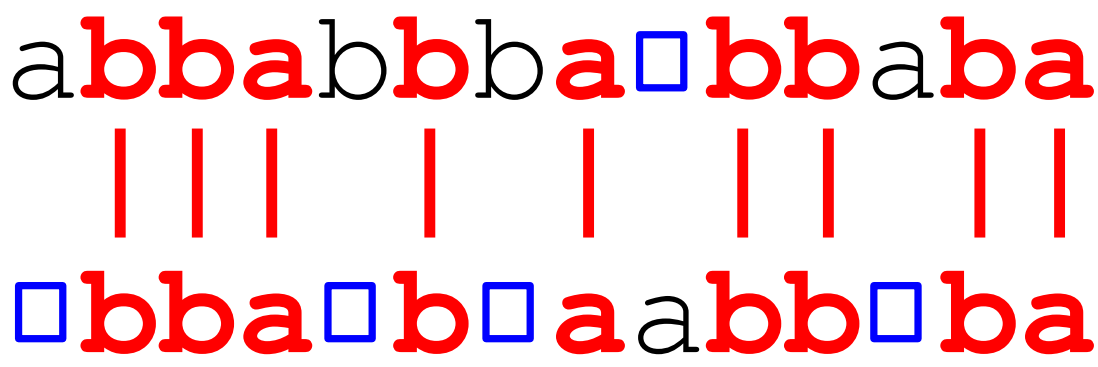
String Alignment

Align the maximum number of letters, preserving order:



String Alignment

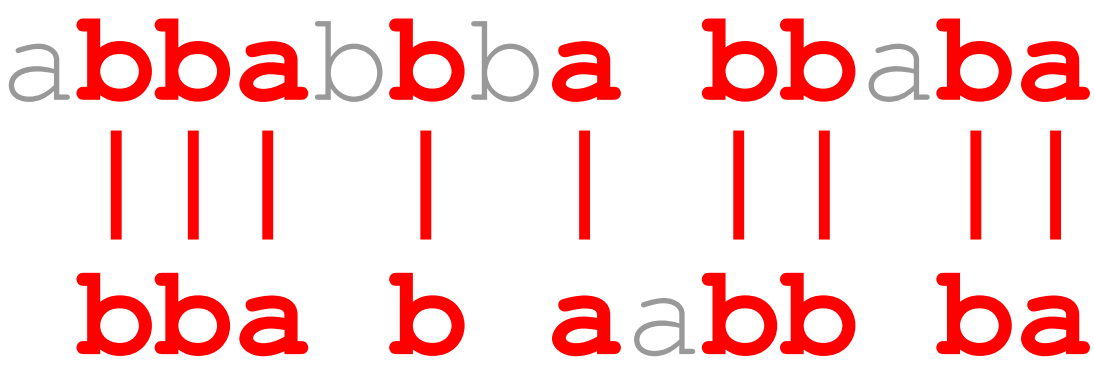
Align the maximum number of letters, preserving order:



□ : inserted gaps

String Alignment

Align the maximum number of letters, preserving order:



The aligned letters form the *longest common subsequence (LCS)*.

String Alignment vs. LCS

The length of the LCS of two strings is a measure for their similarity.

We define the *LCS distance* as:

$$\text{dist}(x, y) = m + n - 2 \cdot |\text{LCS}(x, y)|$$

String Alignment and Edit Distances

Edit distance Minimize the number of *insertions*, *deletions*, and *exchange operations*.

Gapped alignment Match score 1, mismatch score 0, gap penalty -0.5

Weighted alignment Assign weights to aligning each pair of characters from Σ using a *pairwise score matrix*.

$O(n^2)$ Solutions for String Alignment

Longest common subsequence Wagner & Fischer, '74

Global (weighted) alignment Needleman & Wunsch, '70

Local alignment Smith & Waterman '81

Faster but less accurate approaches

BLAST/similar approaches Heuristic search based on frequent DNA substrings to "seed" alignments.

This is very fast!

Less sensitive for aligning regions of low similarity.

$\Rightarrow$ Can miss alignments!

Faster but less accurate approaches

BLAST/similar approaches Heuristic search based on frequent DNA substrings to "seed" alignments.

This is very fast!

Less sensitive for aligning regions of low similarity.

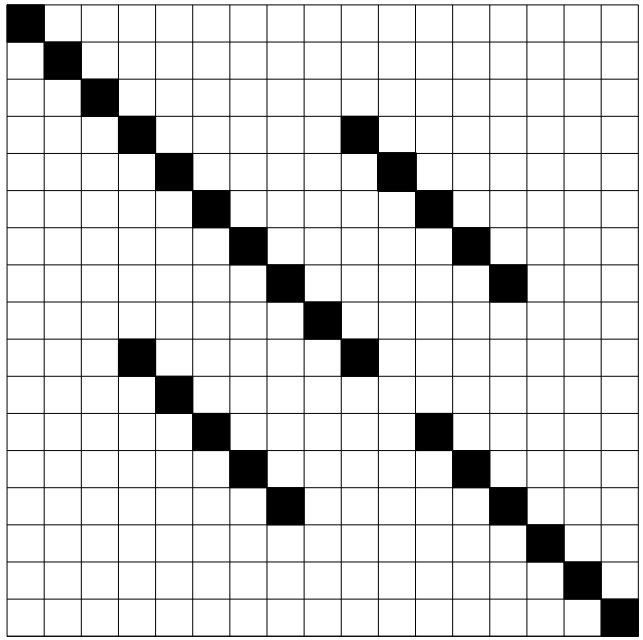
$\Rightarrow$ Can miss alignments!

Faster but less accurate approaches

Dot-plots Compare all substrings of a fixed length w using the Hamming distance.

Plot a point for every window pair scoring above threshold.

$\Rightarrow$ Does not account for gaps!

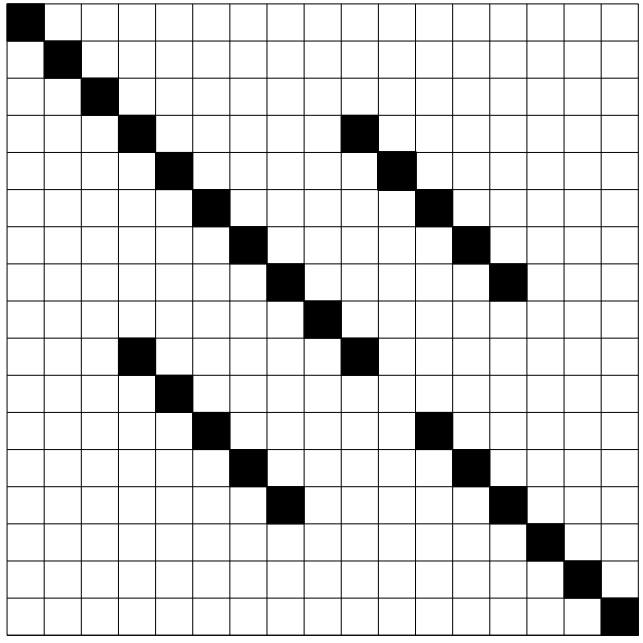


Faster but less accurate approaches

Dot-plots Compare all substrings of a fixed length w using the Hamming distance.

Plot a point for every window pair scoring above threshold.

$\Rightarrow$ Does not account for gaps!



Alignment Plots

Input: Strings x and y, $|x| = m$, $|y| = n$, fixed window length w.

We compare all windows of length w in x to all windows of length w in y (pairwise).

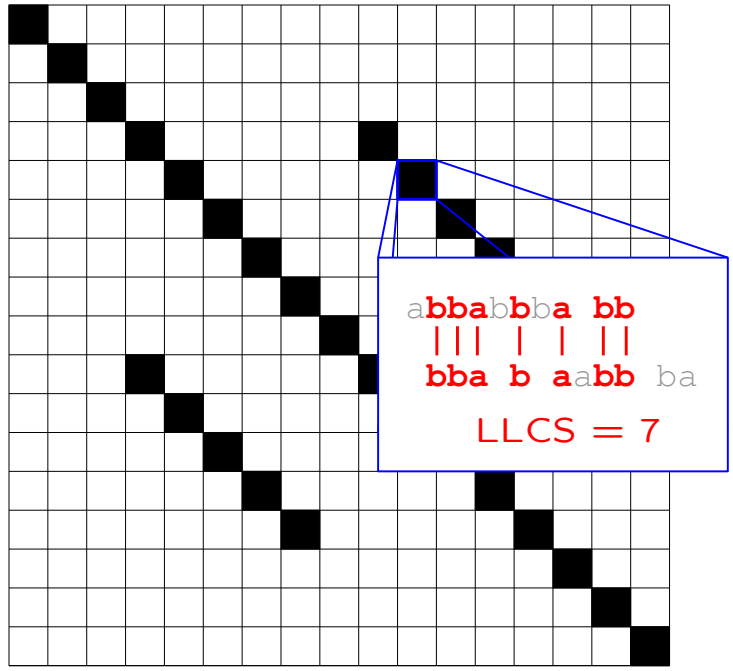
We use a weighted alignment score for comparison.

Computing Alignment Plots

"Naive" algorithm:
Compute scores separately for each pair of windows in $O(mnw^2)$ time.

Heuristic improvements (Ott, 2008): ×25 speedup, same asymptotic running time.

Rasmussen et al., 2004: Efficient algorithm for computing scores for all window pairs with $> 90\%$ similarity.



Computing Alignment Plots

Why? Very sensitive *local* comparison, also for low window similarity (50-70%). Finds things BLAST doesn't.

How big? Input sequences can be very large:
entire genomes should be possible (30MBases – 1TBase)

Window sizes? Typical w-value: around 100.

New Algorithms for Alignment Plots

Algorithmic Improvements We reduce dependency on window size: New practical $O(mn\sqrt{w})$ method.

Vector-Parallelism We can (still) use vector-parallelism.

Parallel Computation Multi-processor computation: running time $O(mn\sqrt{w}/p)$ on p processors.

Algorithmic Tool: Semi-local String Comparison

Definition

Given two strings x and y, compute *highest-score matrix* A with

$$A(i, j) = |LCS(x, y_i \ldots y_j)|.$$

We compare all substrings in y to entire string x.

Algorithm [Schmidt:98,Alves+:06]

We can compute A in $O(n^2)$ time.

Implicit Highest-Score Matrices

Theorem (Tiskin:05)

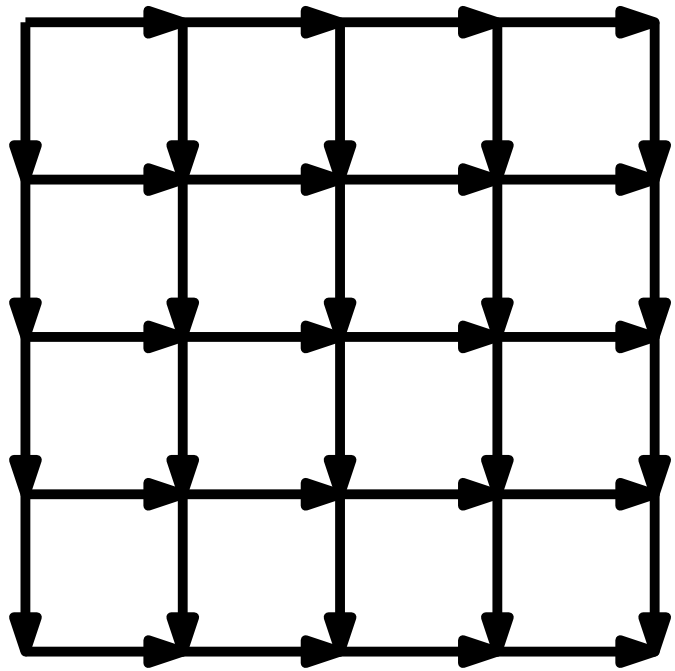
The highest-score matrix for comparing x and y can be represented by a permutation of size $O(m+n)$.

Seaweed Algorithm

We can compute this permutation incrementally by dynamic programming in $O(mn)$.

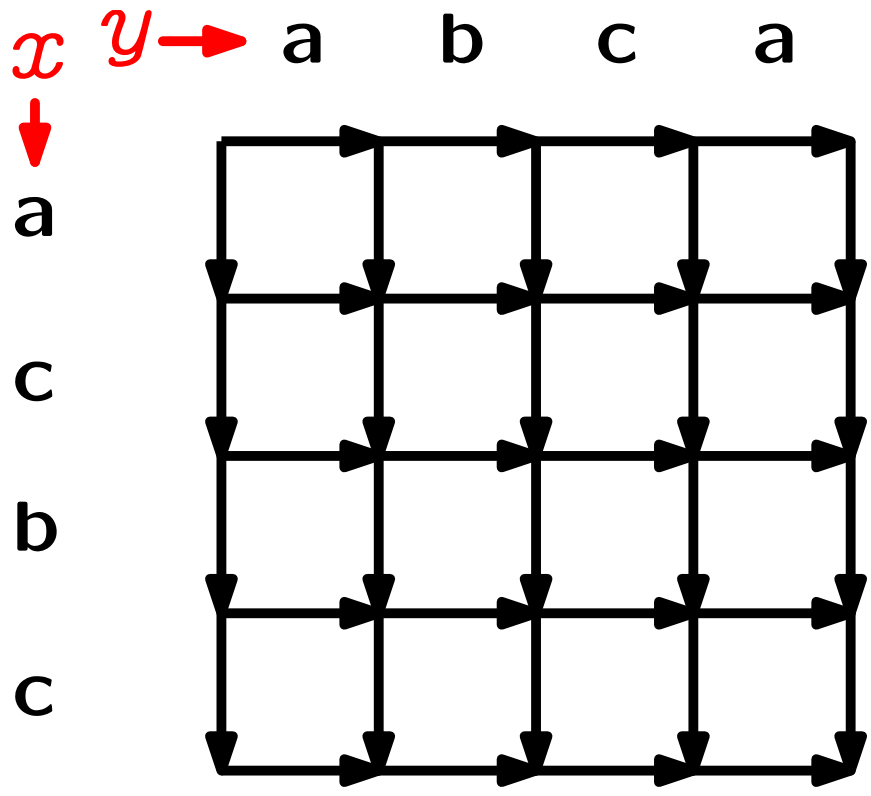
The Seaweed Algorithm

We draw the *alignment-dag*...



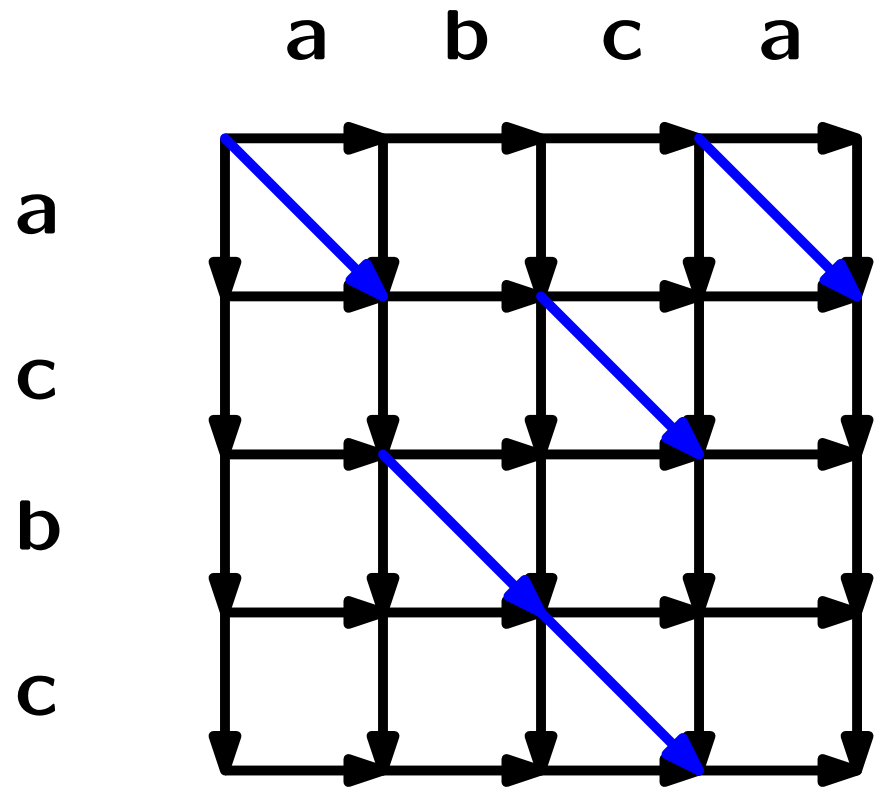
The Seaweed Algorithm

. . . that corresponds to the input strings.



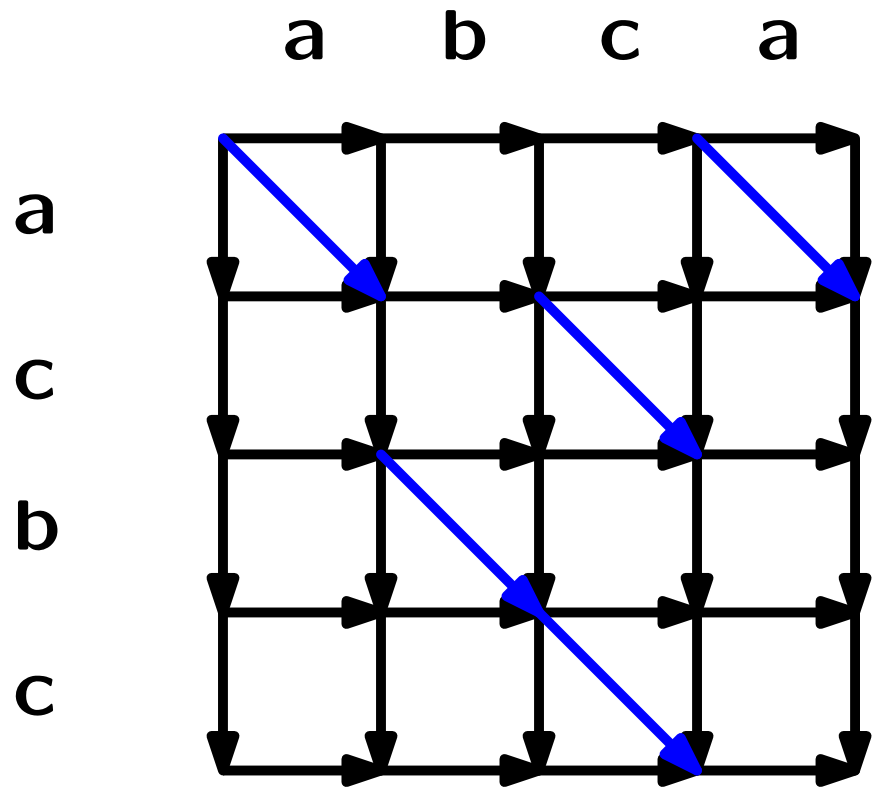
The Seaweed Algorithm

We insert blue edges for every match.



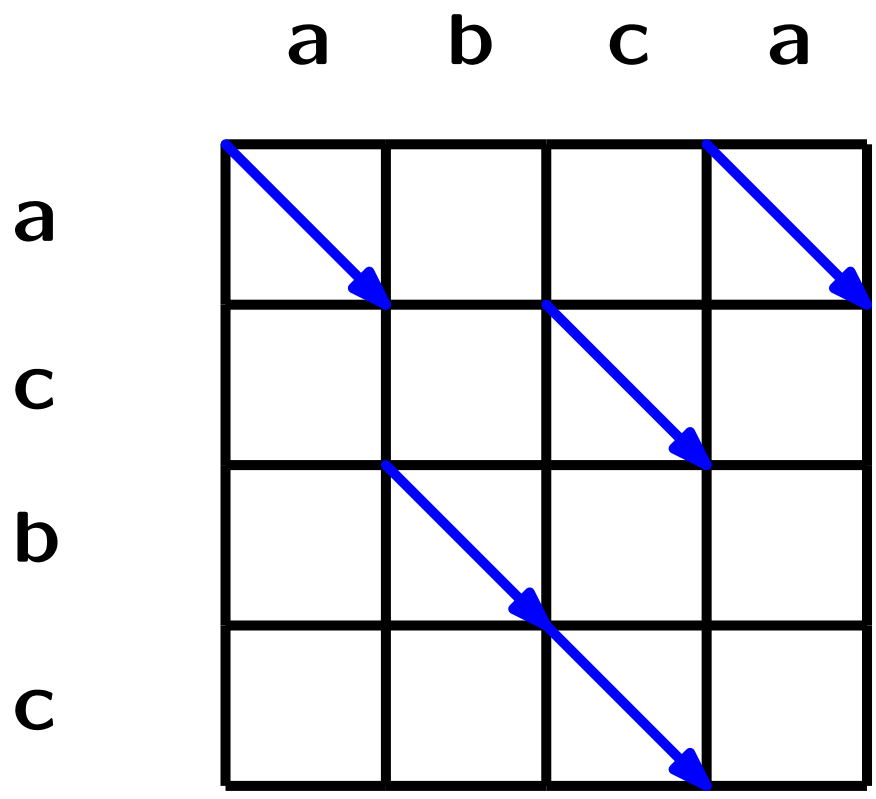
The Seaweed Algorithm

Blue edges have weight 1.



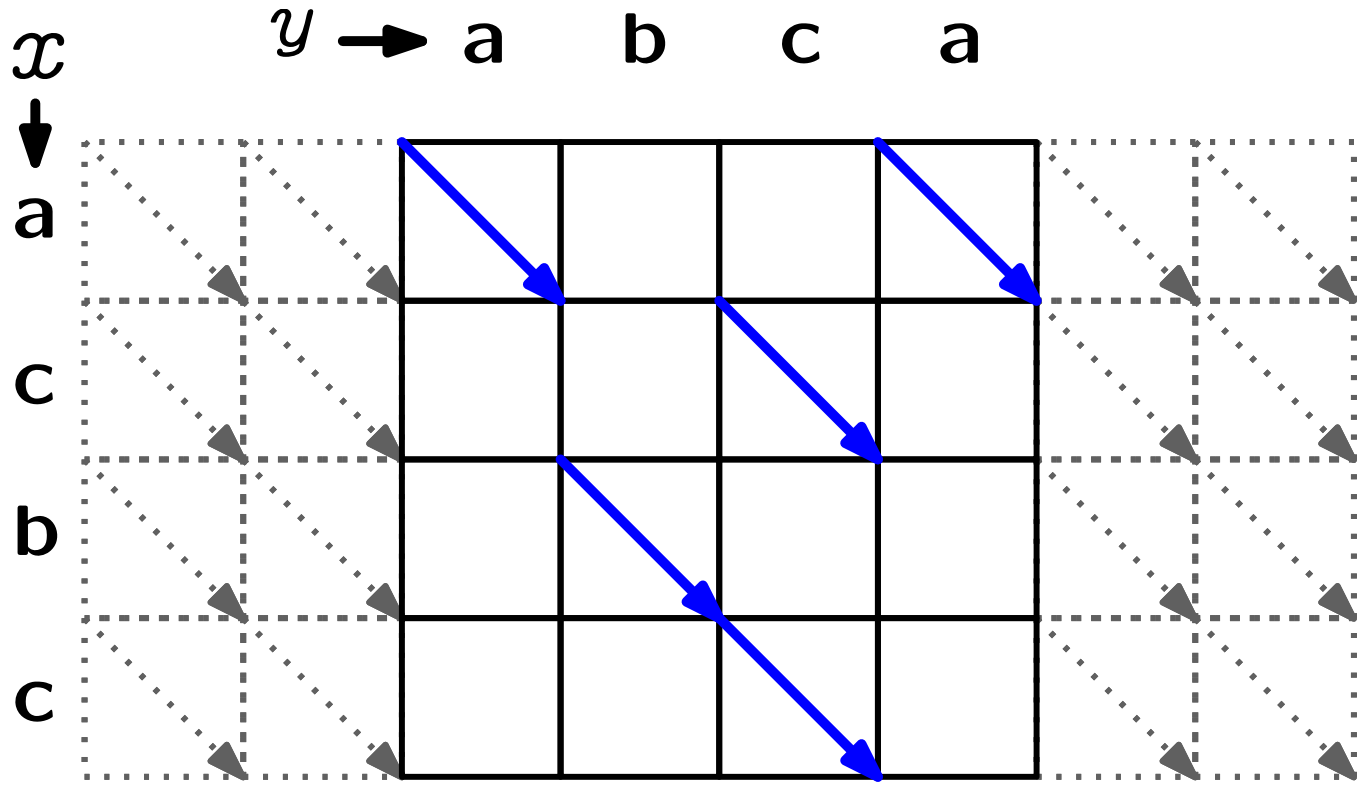
The Seaweed Algorithm

Black edges have weight 0.



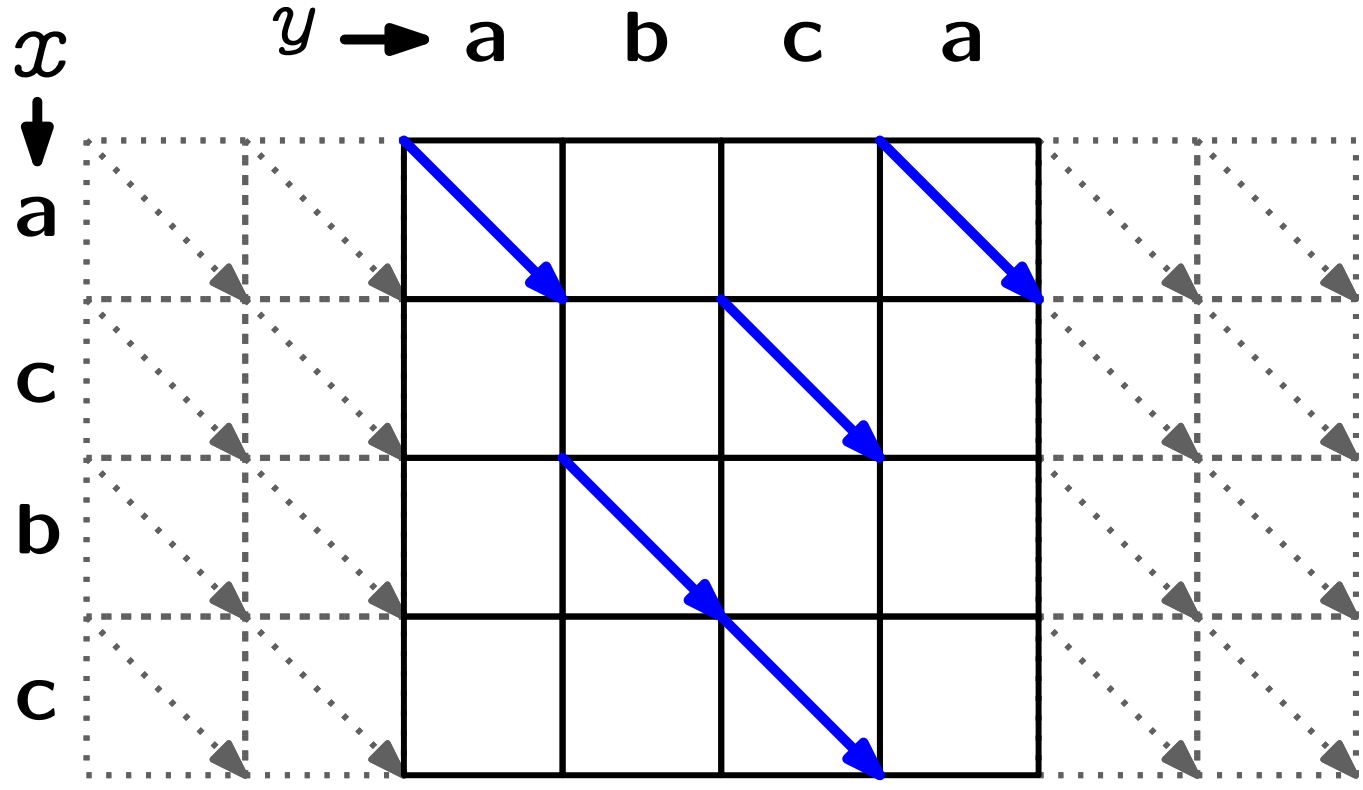
The Seaweed Algorithm

We can extend the dag with matches to the left and right.



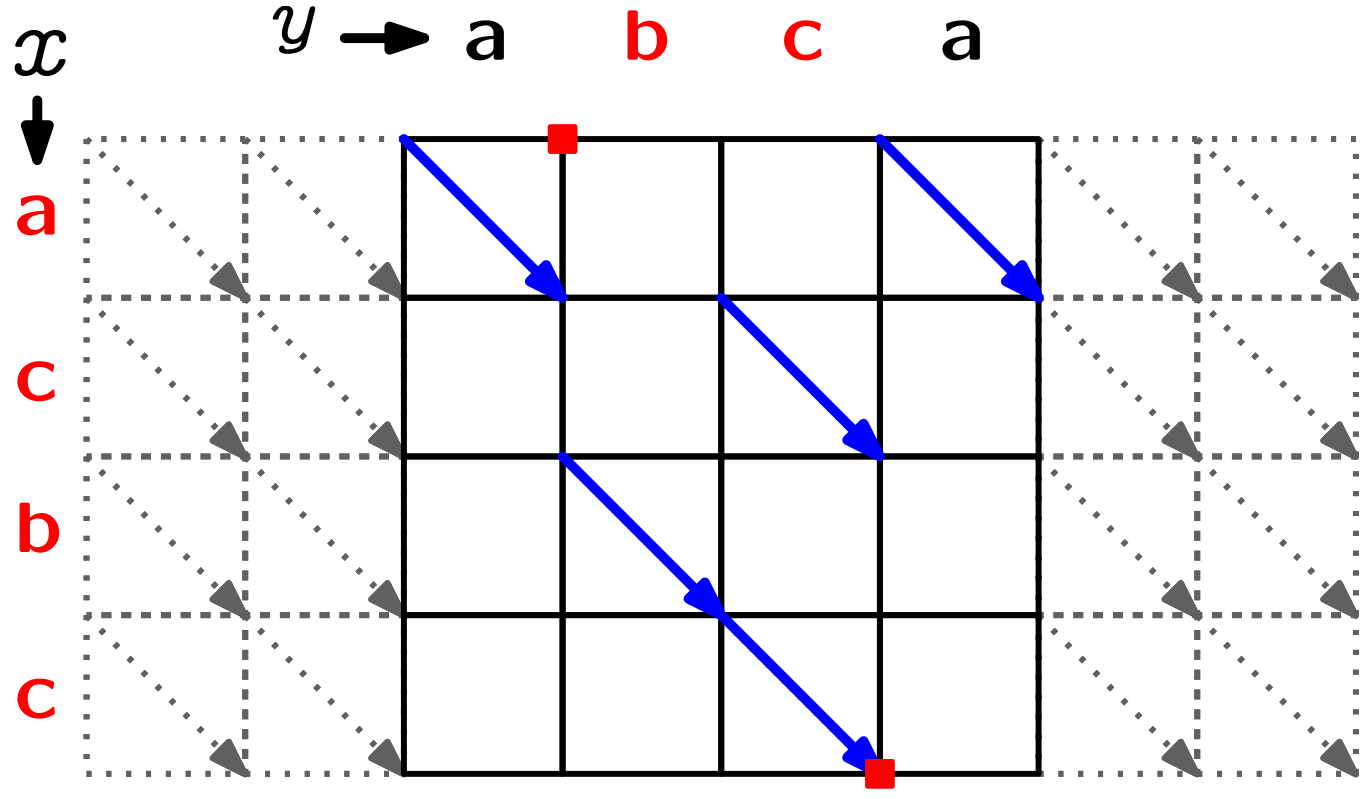
The Seaweed Algorithm

Drawing this dag partitions the plane into cells.



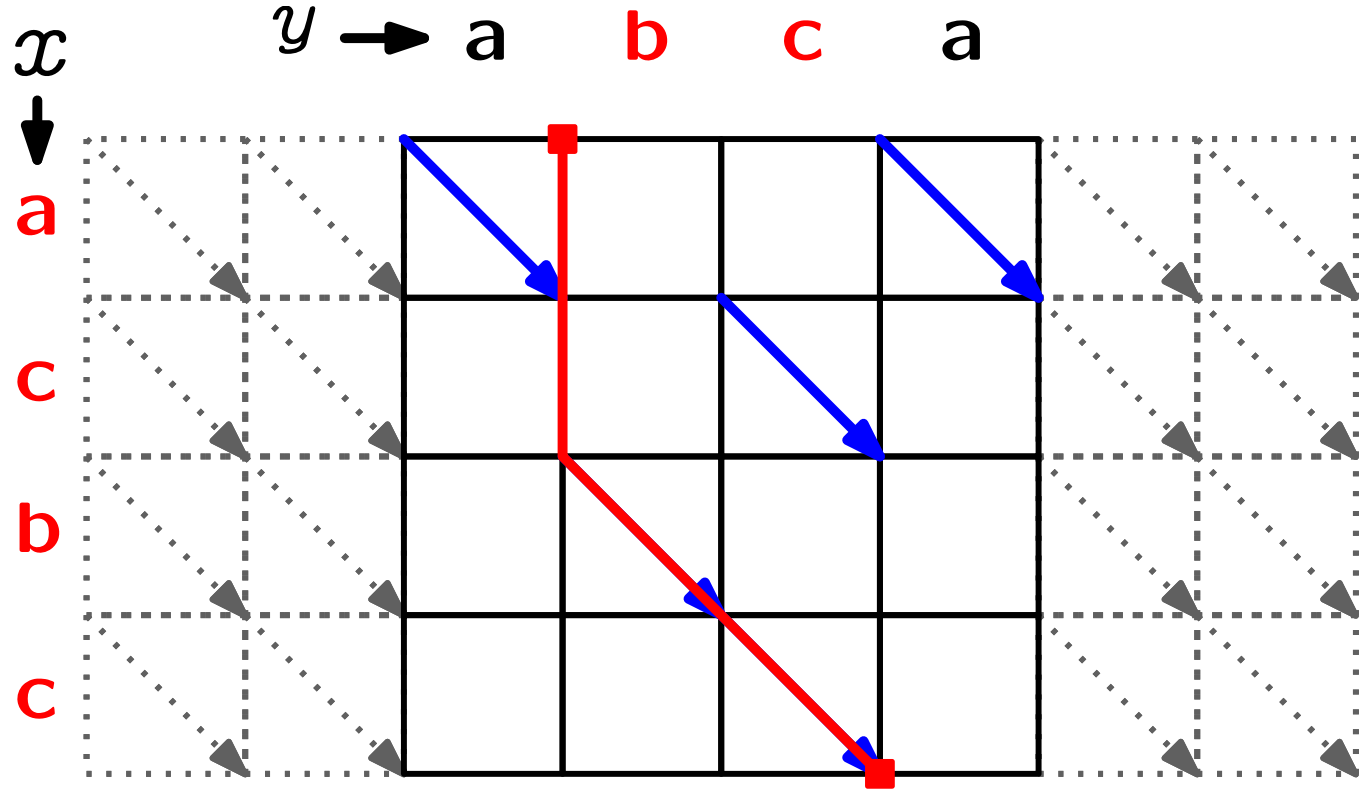
The Seaweed Algorithm

Alignment lengths in $A(i, j)$ correspond to longest paths.



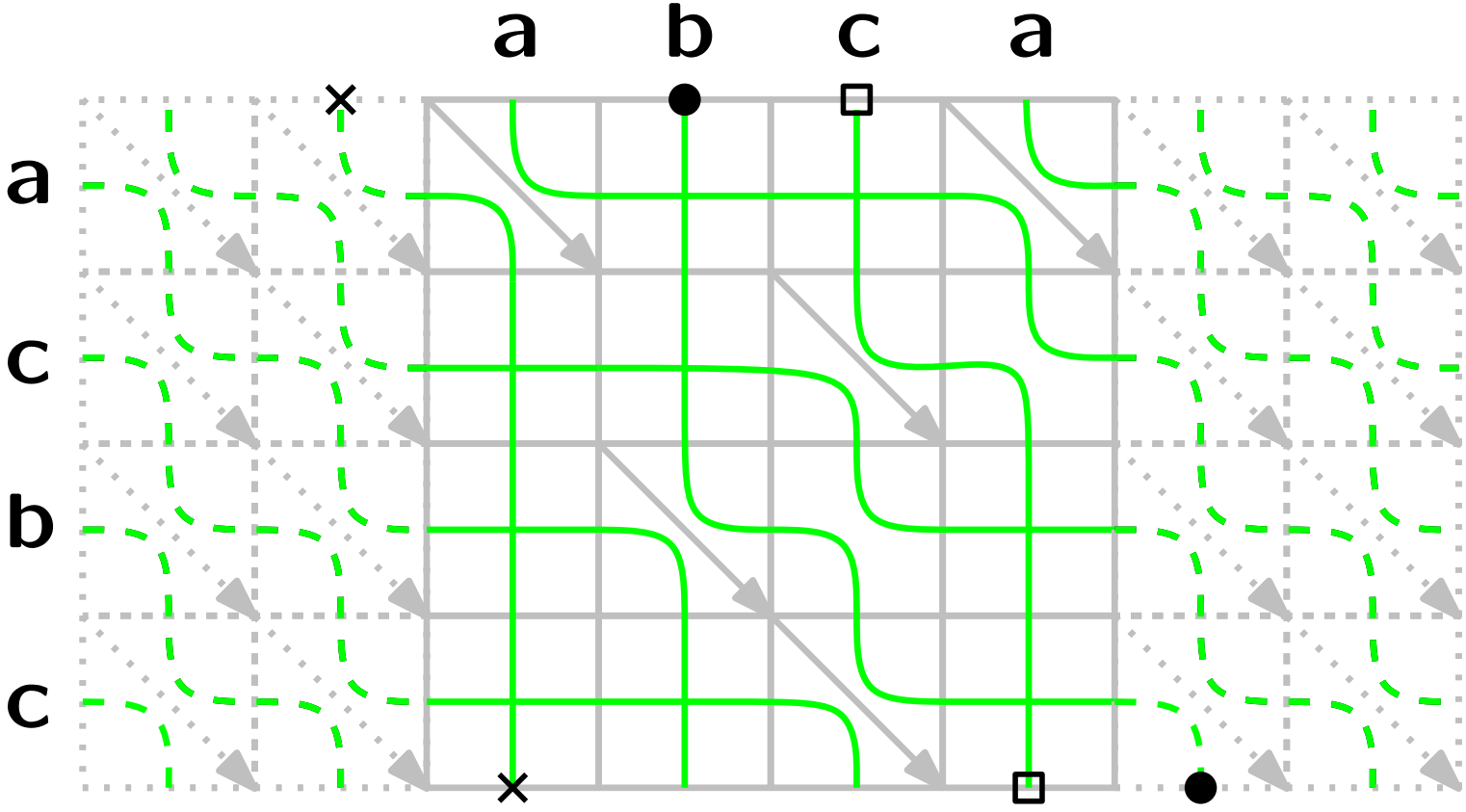
The Seaweed Algorithm

Alignment lengths in $A(i, j)$ correspond to longest paths.



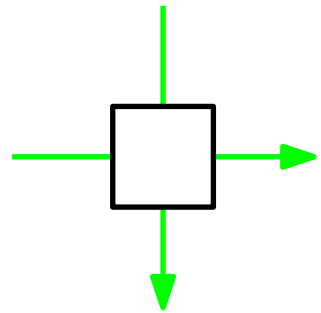
The Seaweed Algorithm

We compute the lengths of these paths implicitly by tracing *seaweeds*.



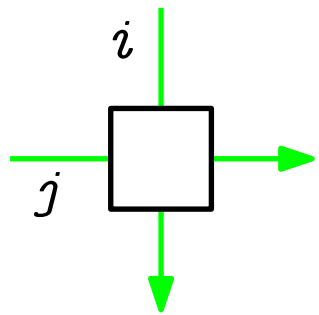
The Seaweed Algorithm

We trace seaweeds through cells.



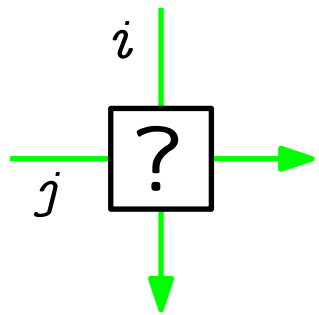
The Seaweed Algorithm

We trace seaweed start and end points.



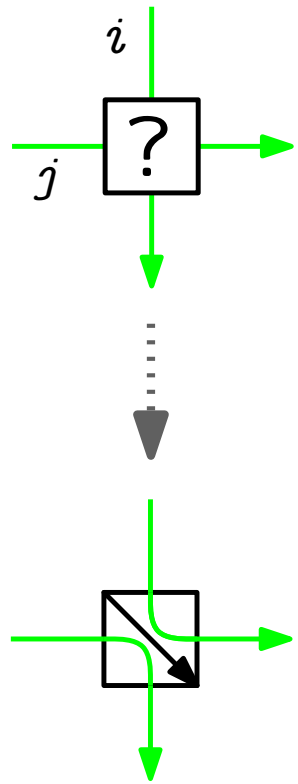
The Seaweed Algorithm

In a cell, seaweeds may or may not cross.



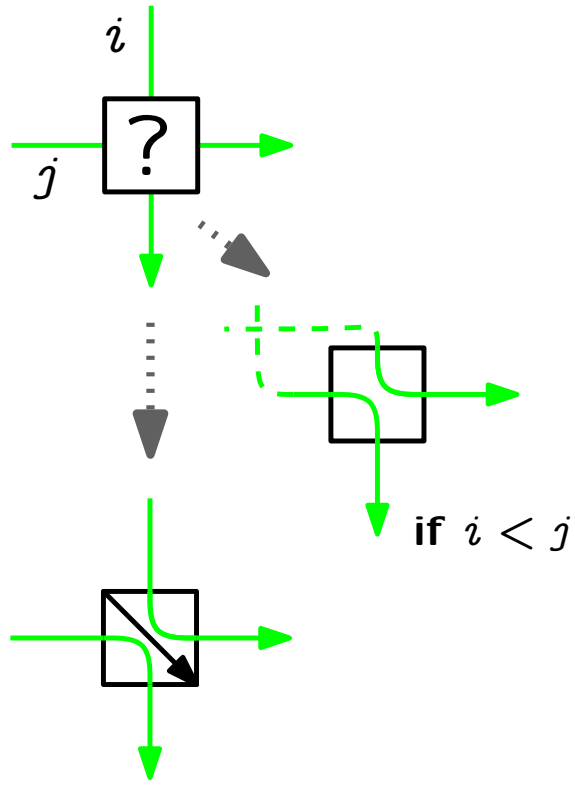
The Seaweed Algorithm

Seaweeds don't cross in match cells.



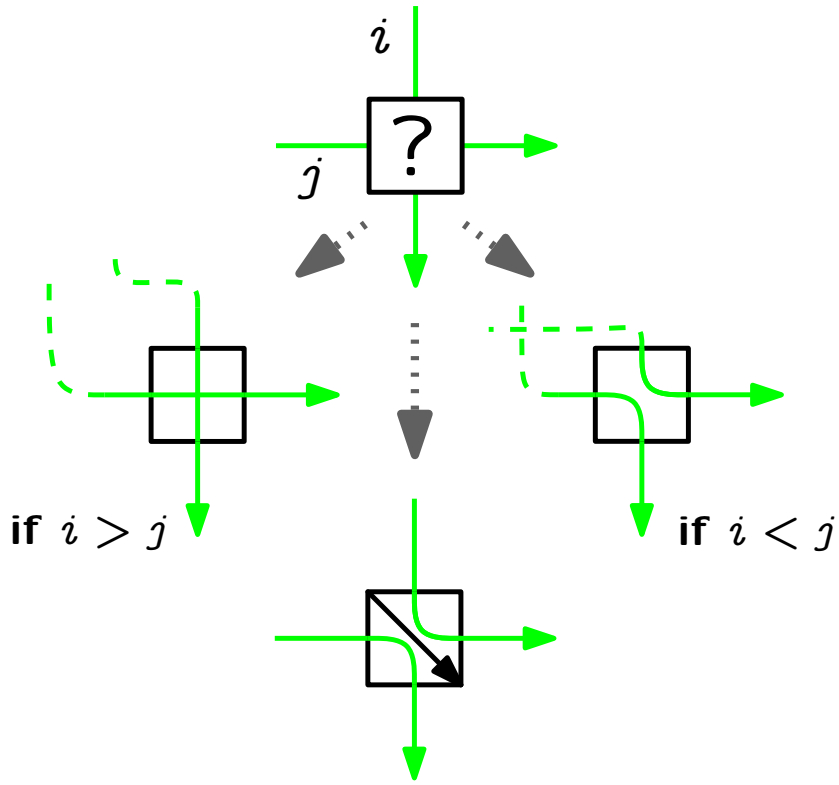
The Seaweed Algorithm

Two seaweeds are allowed to cross at most once.



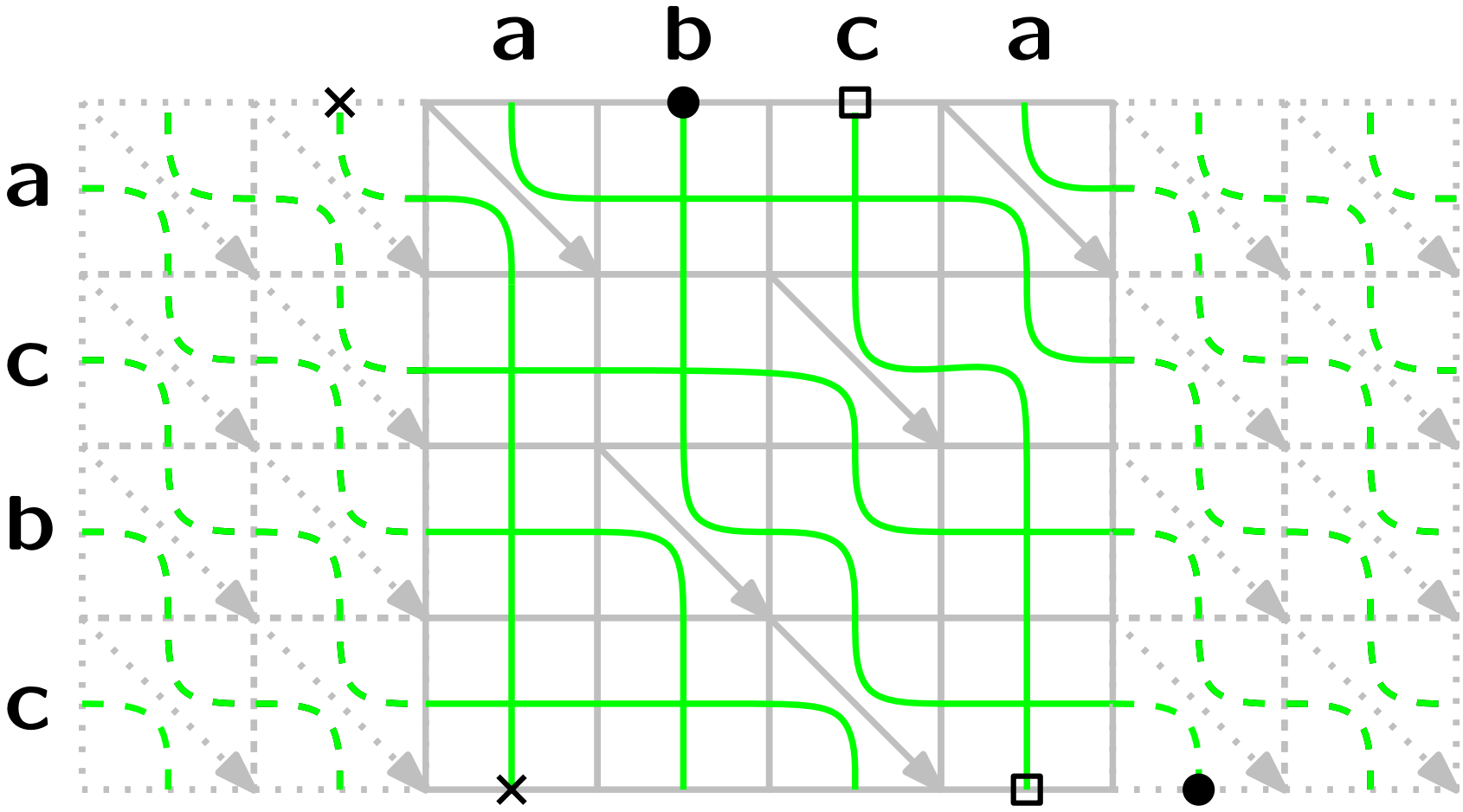
The Seaweed Algorithm

Two seaweeds are allowed to cross at most once.



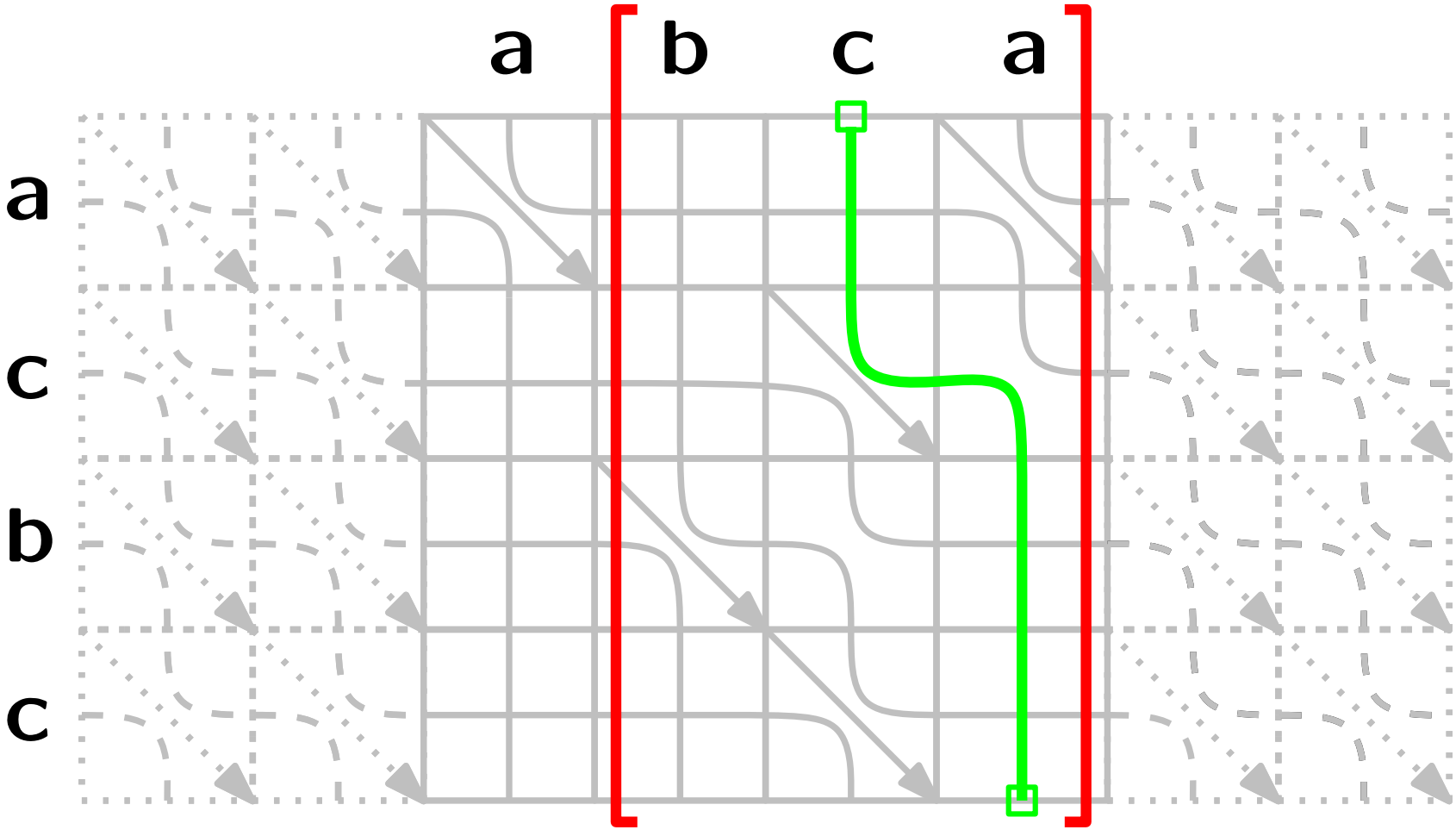
Querying the LCS Distance

Given all seaweeds. . .



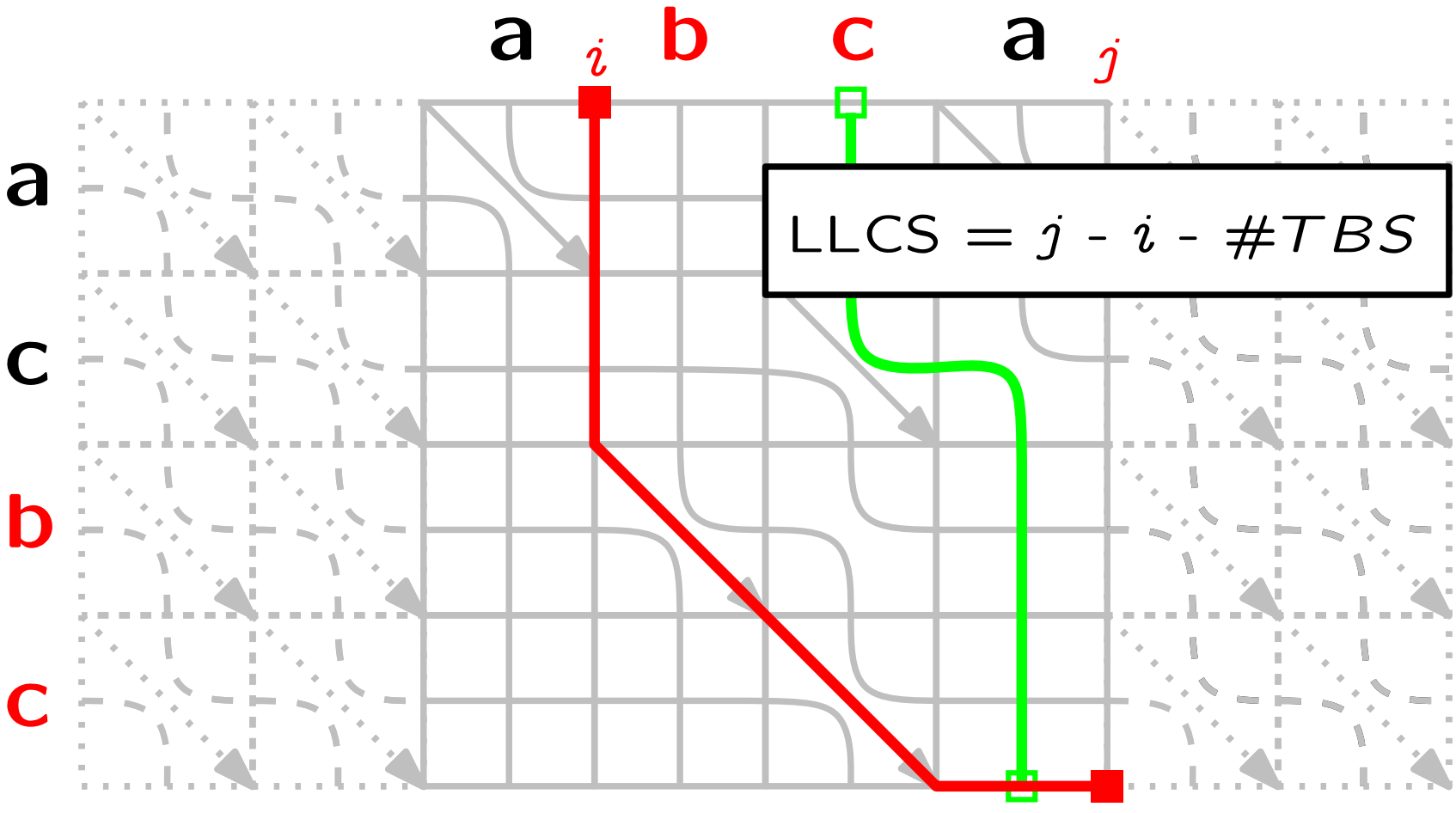
Querying the LCS Distance

. . . we can count seaweeds in an interval:



Querying the LCS Distance

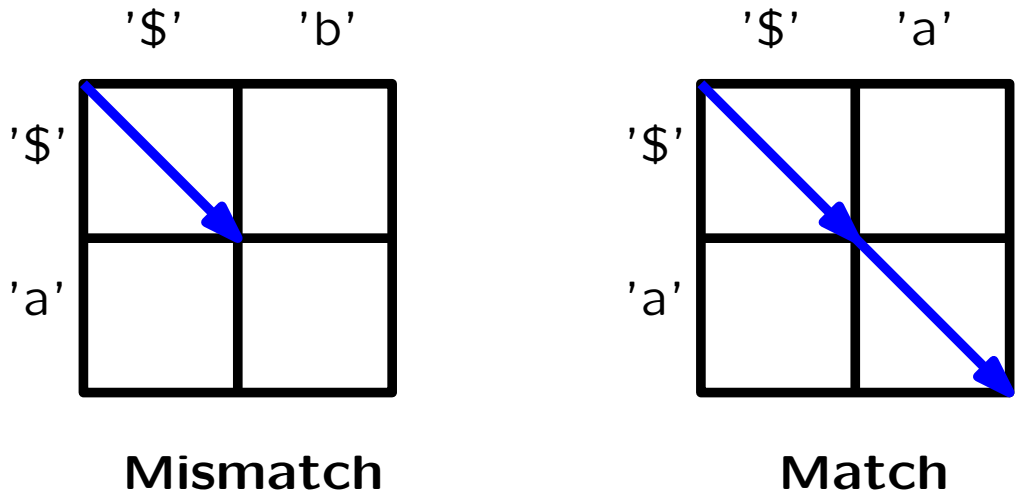
. . . and obtain the LLCS:



Rational Scores using Seaweeds

We can deal with rational pairwise score matrices (constant factor slowdown).

$$w_{=} = 1$$
$$w_{\neq} = 0$$
$$w_{\sqcup} = -0.5$$

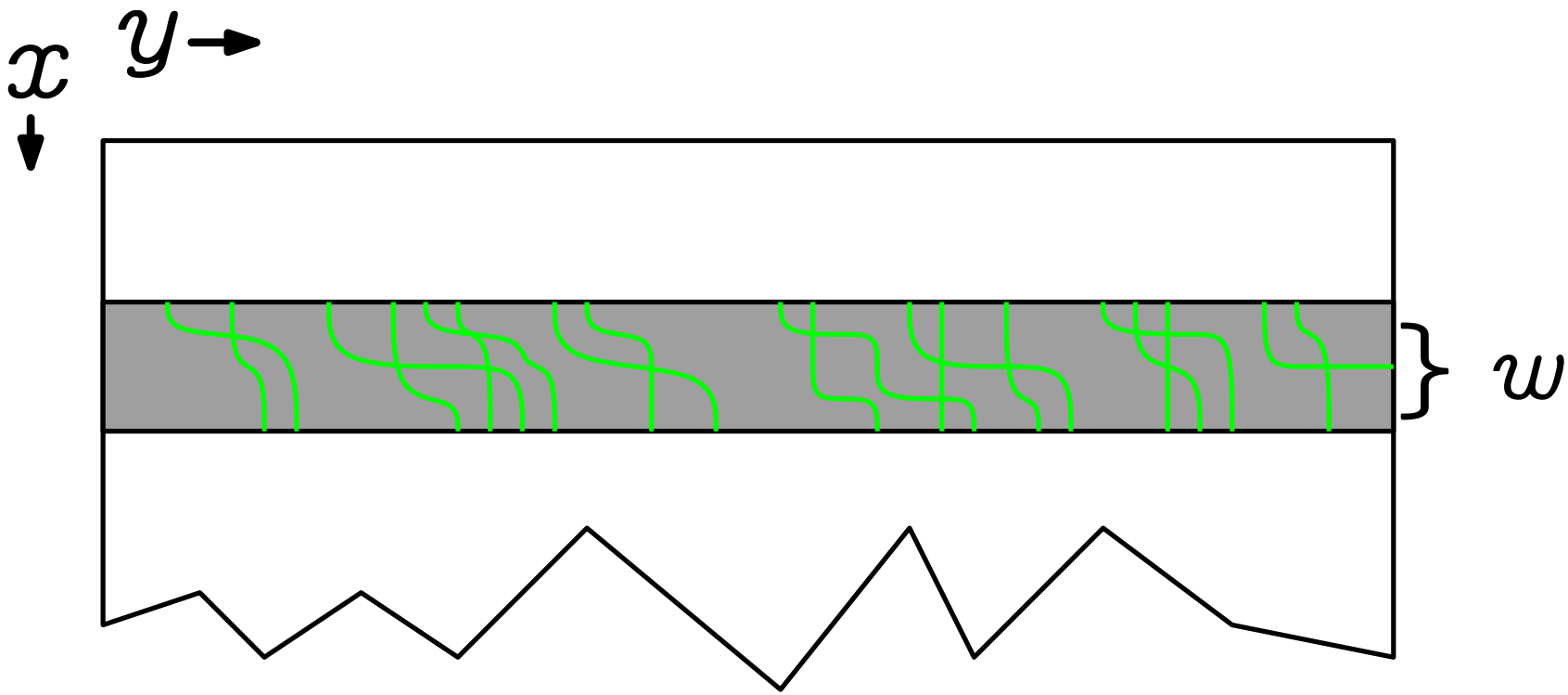


$$S(x, y) = LLCS(x', y') - 0.5 \cdot (m + n)$$

Outline

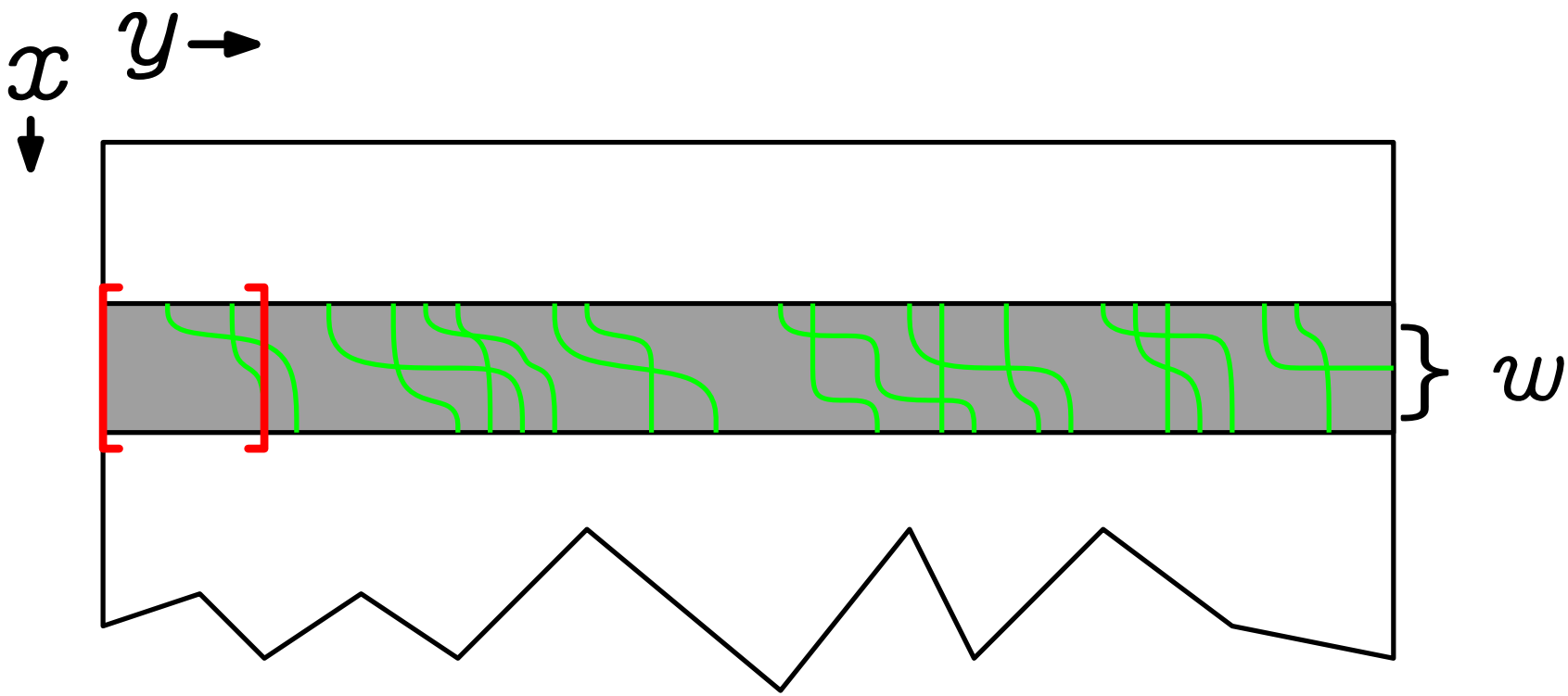
Computing Alignment Plots Using Seaweeds

We compute seaweeds for y against all substrings of x with length w.



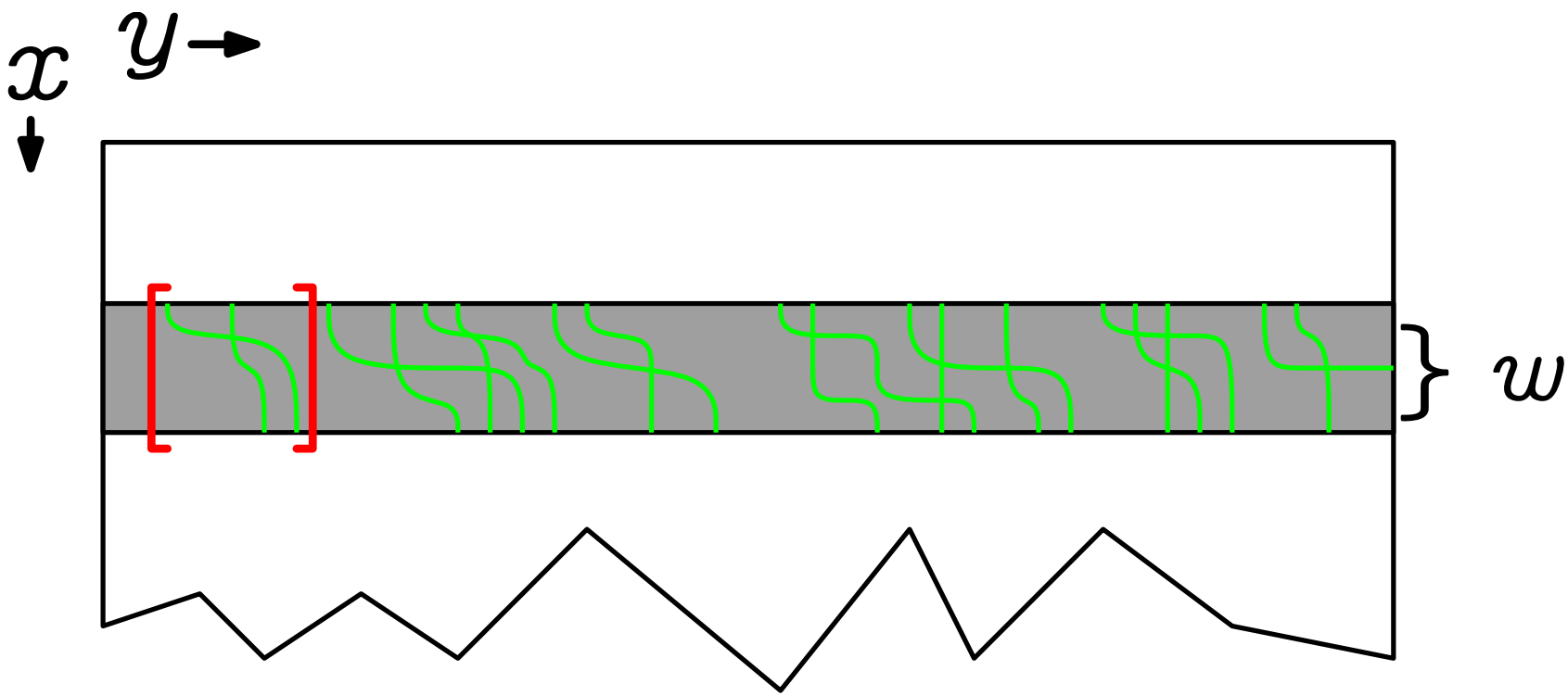
Computing Alignment Plots Using Seaweeds

Inside each *strip*, we count seaweeds within a sliding w-window.



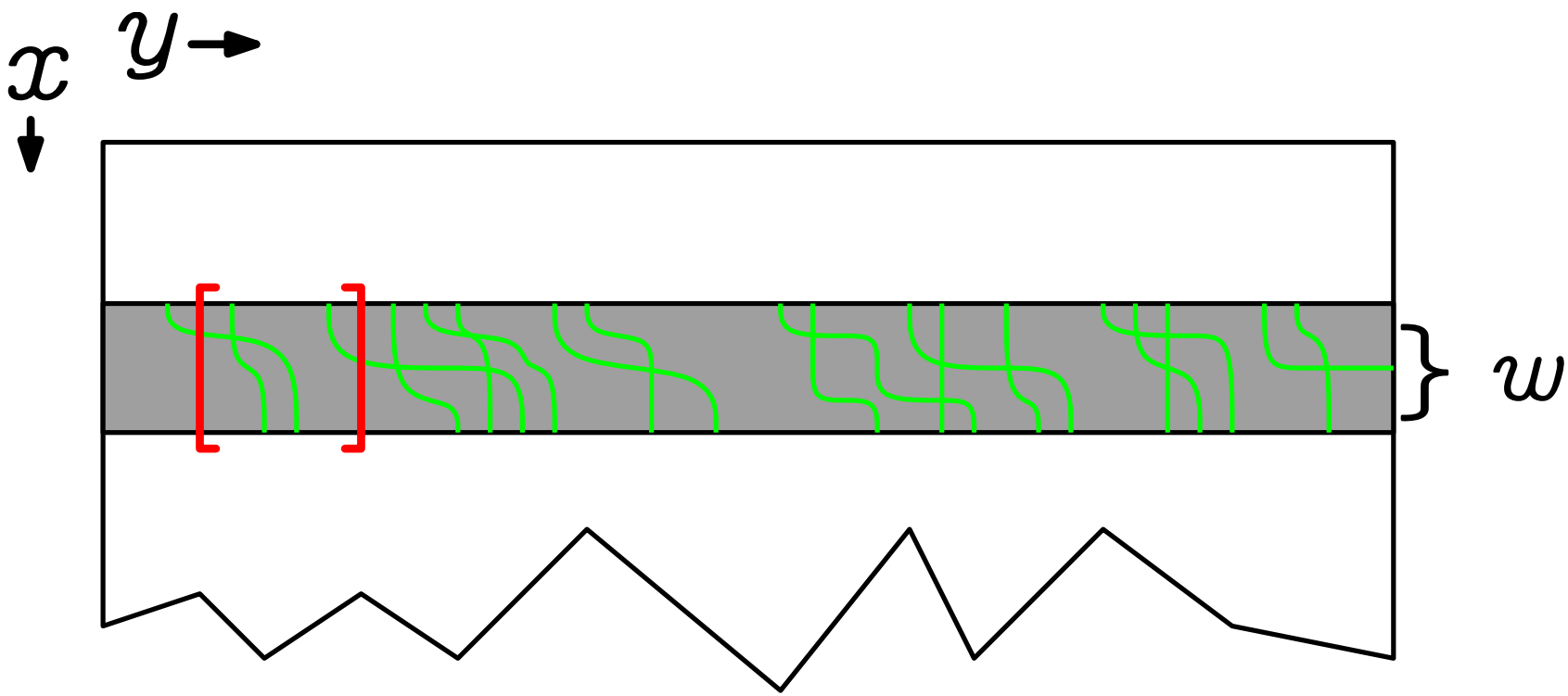
Computing Alignment Plots Using Seaweeds

Inside each *strip*, we count seaweeds within a sliding w-window.



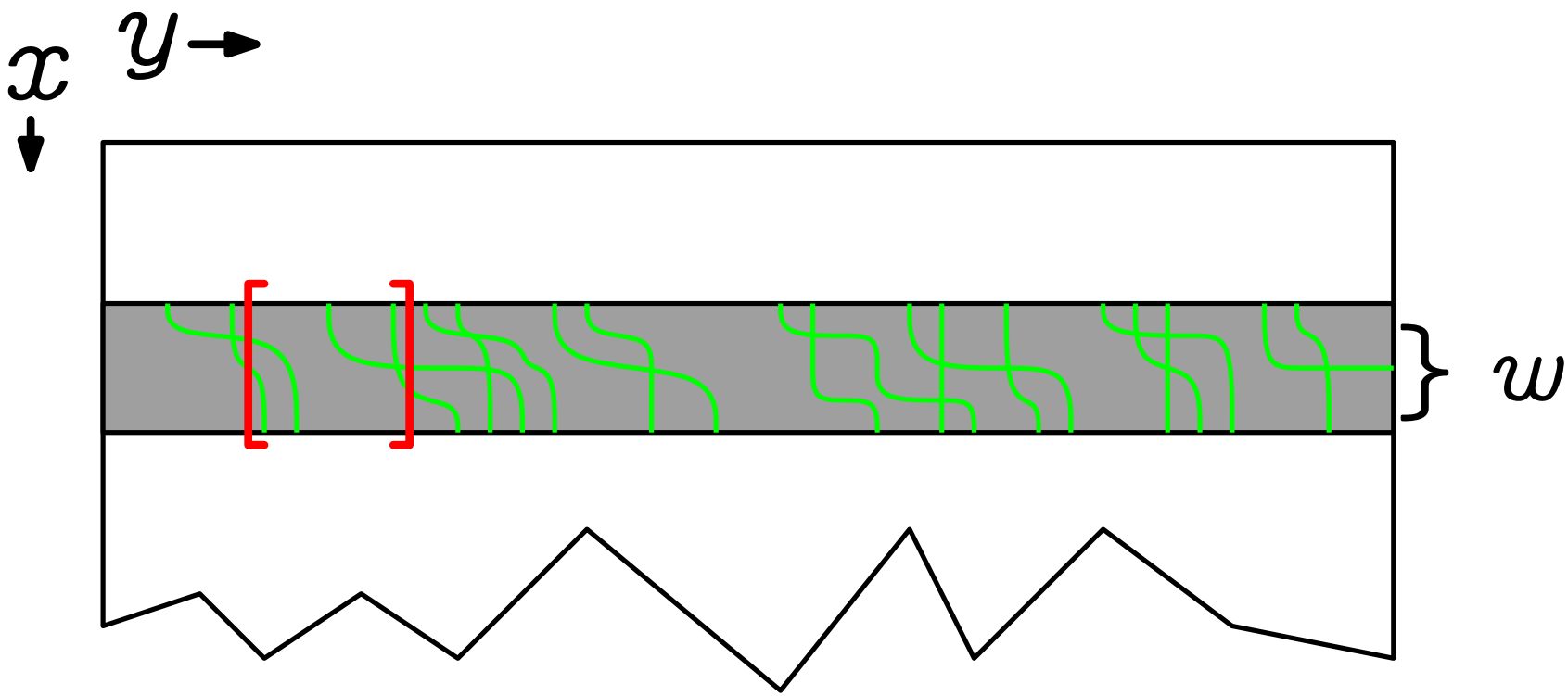
Computing Alignment Plots Using Seaweeds

Inside each *strip*, we count seaweeds within a sliding w-window.



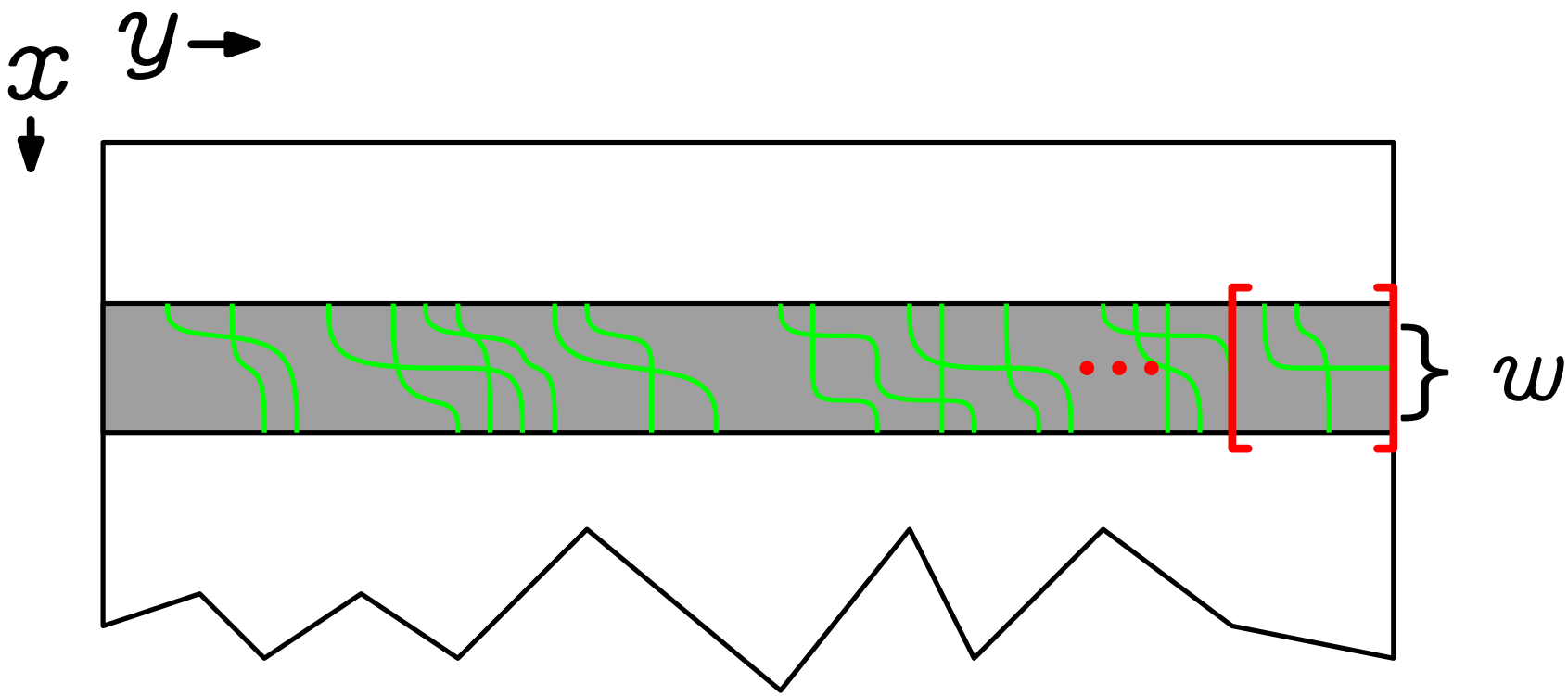
Computing Alignment Plots Using Seaweeds

Inside each *strip*, we count seaweeds within a sliding w-window.



Computing Alignment Plots Using Seaweeds

Inside each *strip*, we count seaweeds within a sliding w-window.



Computing Alignment Plots Using Seaweeds

We have $m - w + 1$ strips, each strip takes time $O(nw)$ to process.

$\Rightarrow$ We get running time $O(mnw)$.

This is not yet optimal – the strips overlap!

How can we improve this?

Computing Alignment Plots Using Seaweeds

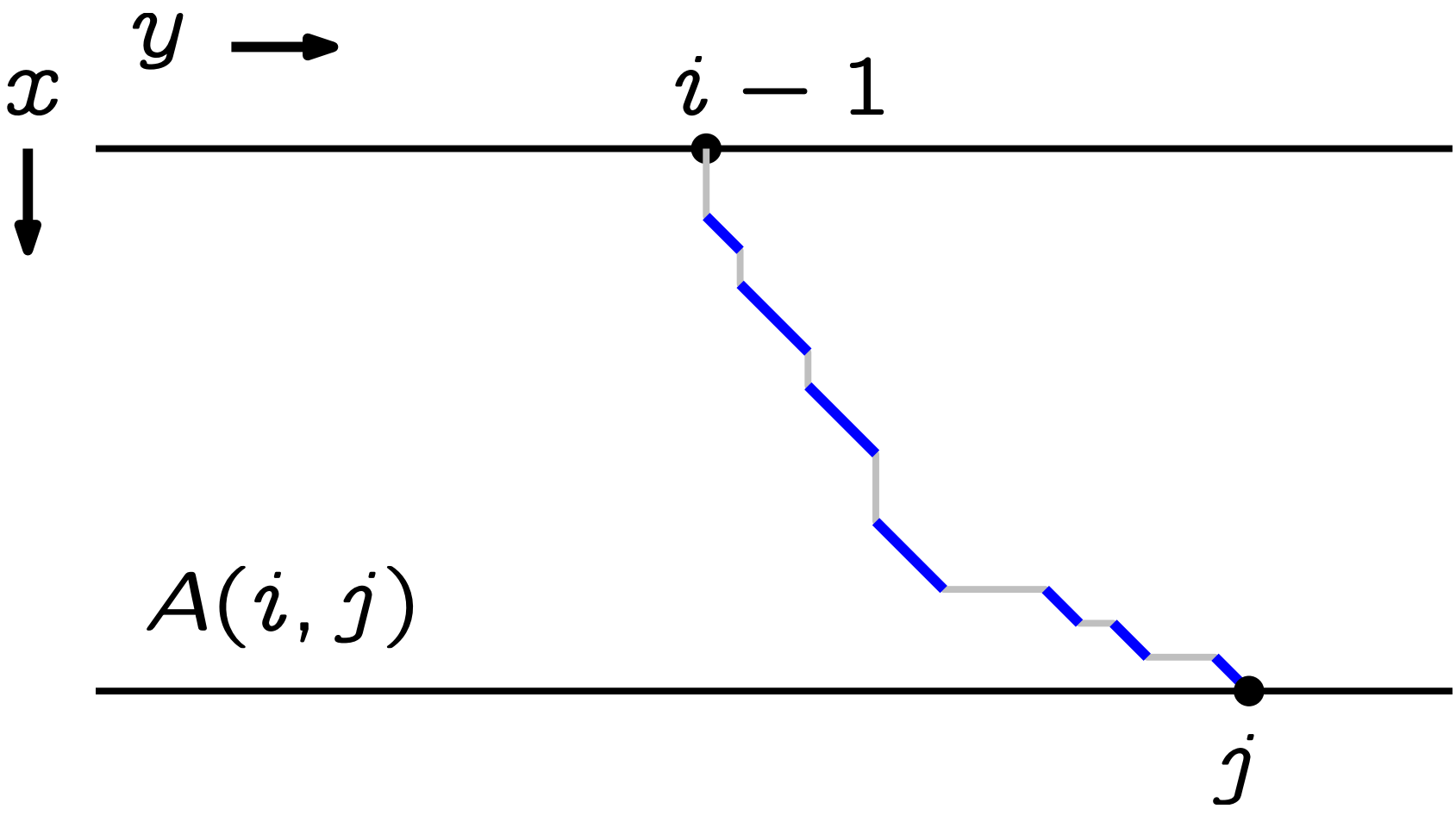
We have $m - w + 1$ strips, each strip takes time $O(nw)$ to process.

$\Rightarrow$ We get running time $O(mnw)$.

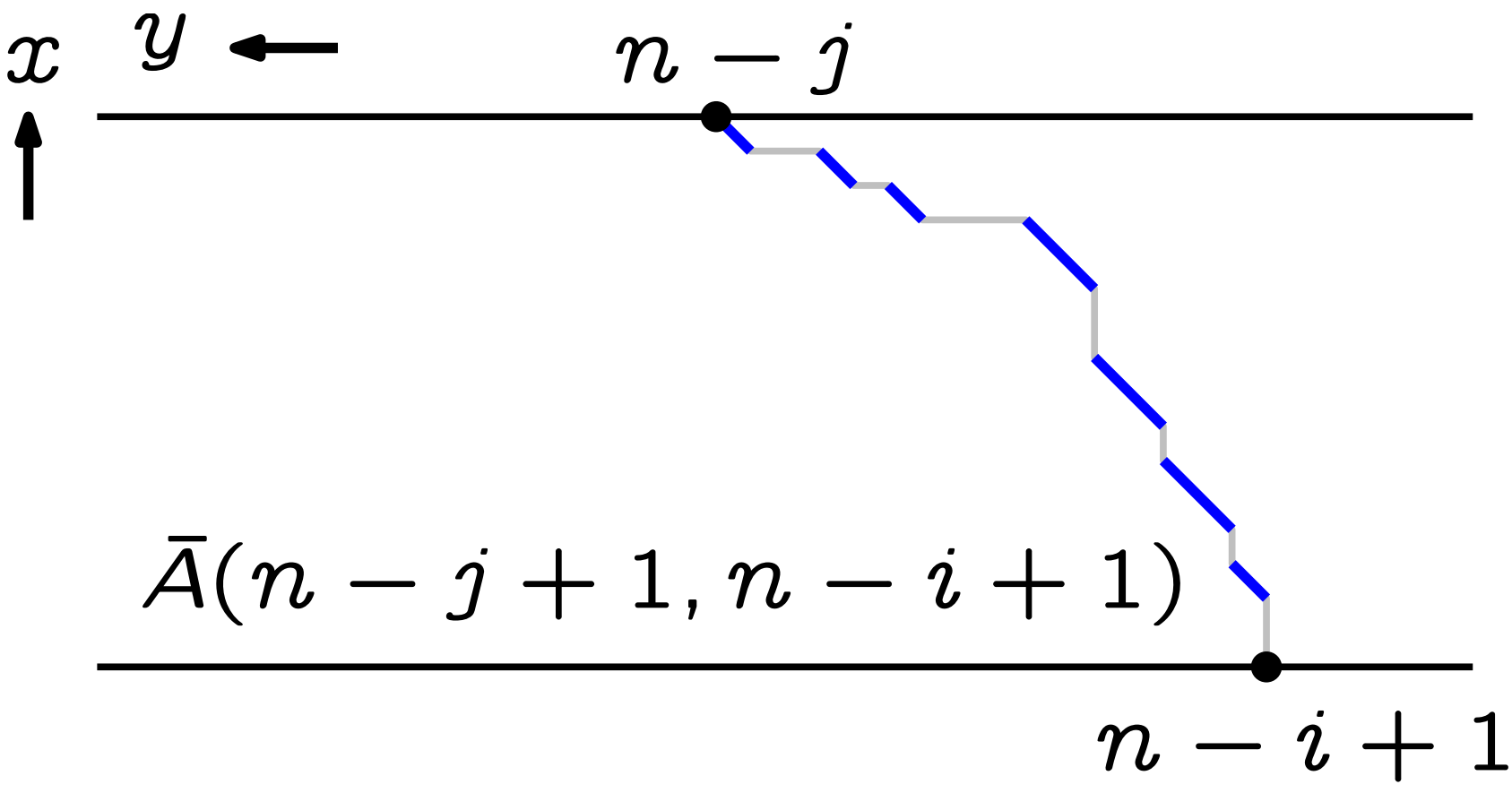
This is not yet optimal – the strips overlap!

How can we improve this?

Extending Strips Upwards



Extending Strips Upwards



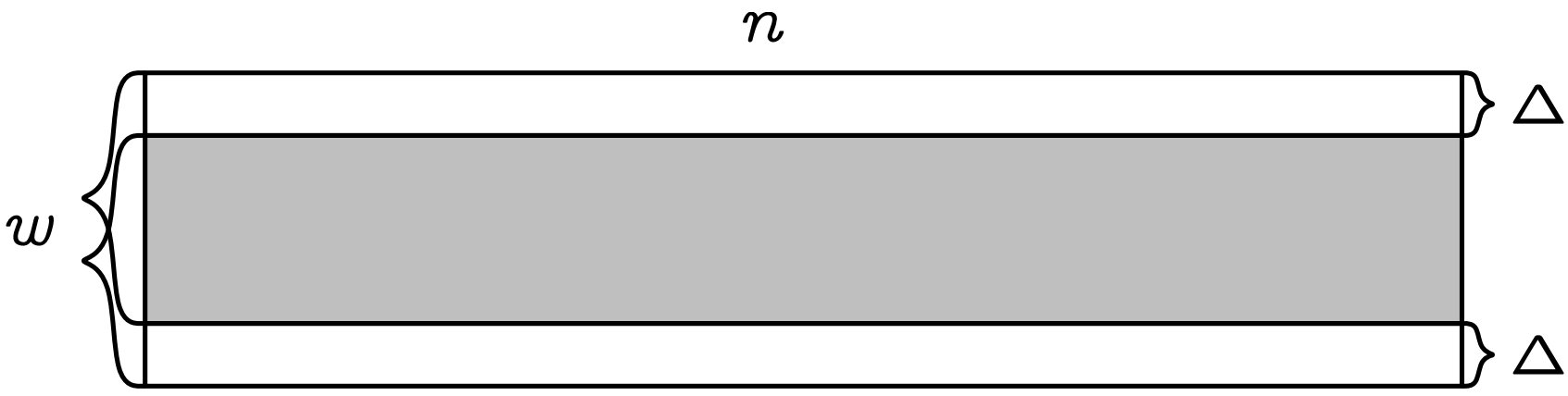
Extending Strips Upwards

Theorem (KT:2009)

Given the permutation corresponding to highest-score matrix A for comparing x and y, we can obtain the permutation for highest-score matrix $\bar{A}$ comparing the reversals $\bar{x}$ and $\bar{y}$ in time $O(m+n)$.

Computing Alignment Plots Using Seaweeds

We can now achieve some speedup by re-using overlapping parts of strips.

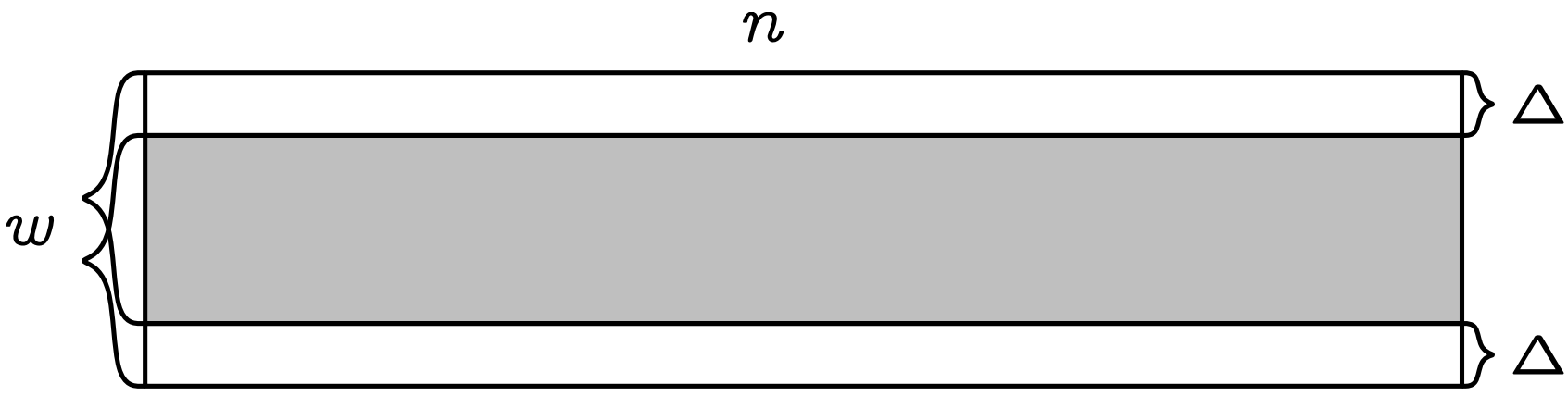


Computing strip seaweeds separately: $2nw$ operations.

Computing by extending the overlapping area:
$n(w - \Delta) + 2n\Delta = n(w + \Delta)$ operations.

Computing Alignment Plots Using Seaweeds

We can now achieve some speedup by re-using overlapping parts of strips.

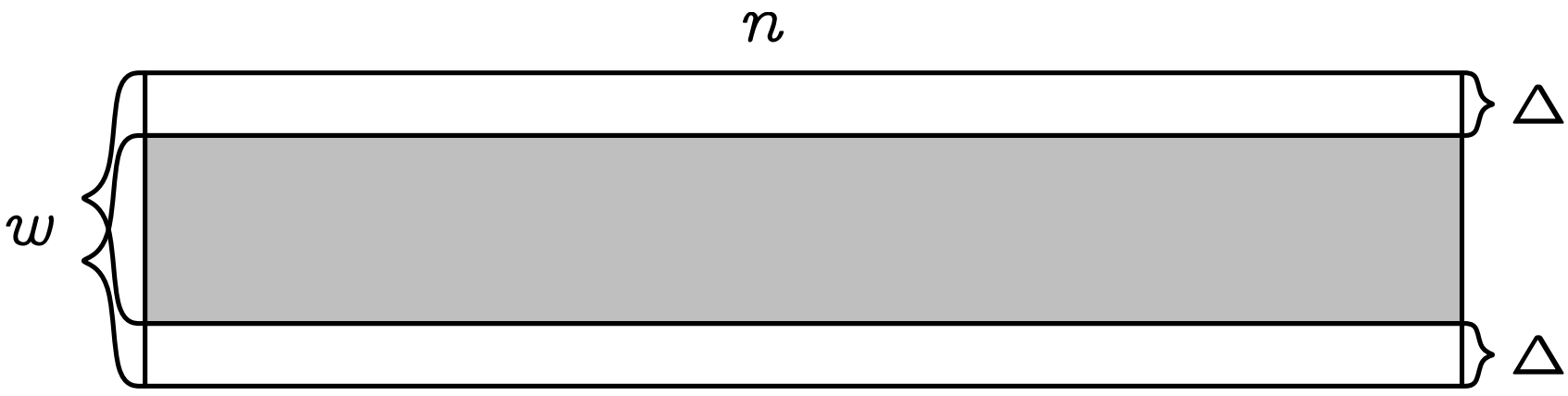


Computing strip seaweeds separately: $2nw$ operations.

Computing by extending the overlapping area: $n(w - \Delta) + 2n\Delta = n(w + \Delta)$ operations.

Computing Alignment Plots Using Seaweeds

We can now achieve some speedup by re-using overlapping parts of strips.

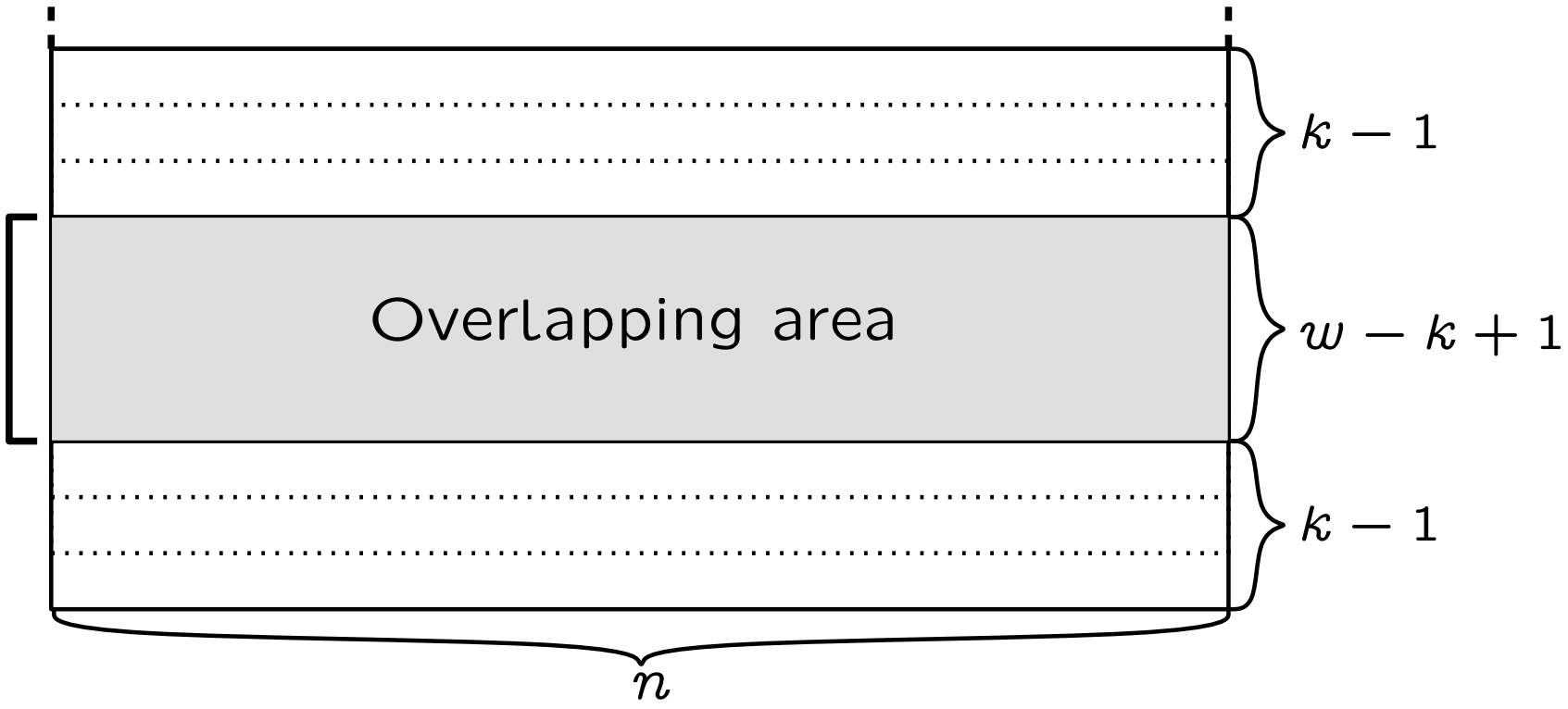


Computing strip seaweeds separately: $2nw$ operations.

Computing by extending the overlapping area:
$n(w - \Delta) + 2n\Delta = n(w + \Delta)$ operations.

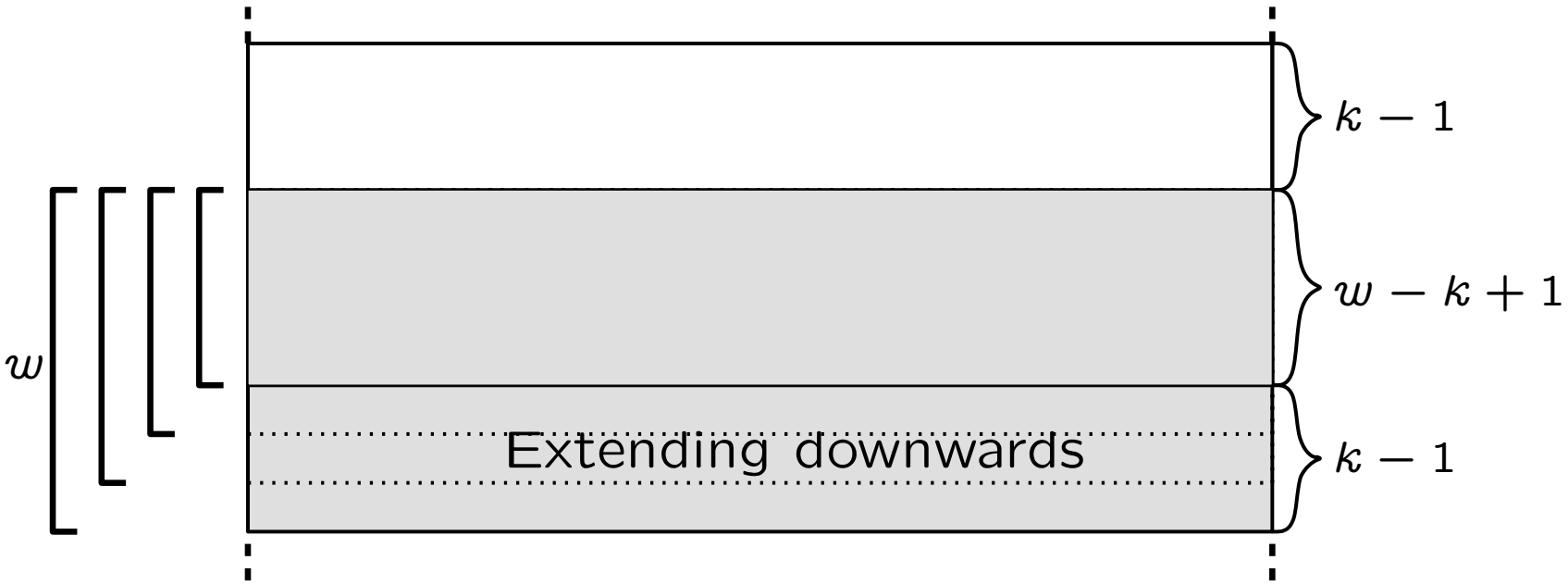
Computing Alignment Plots Using Seaweeds

Step 1: Work $c_1(n) = n \cdot (w - k + 1)$



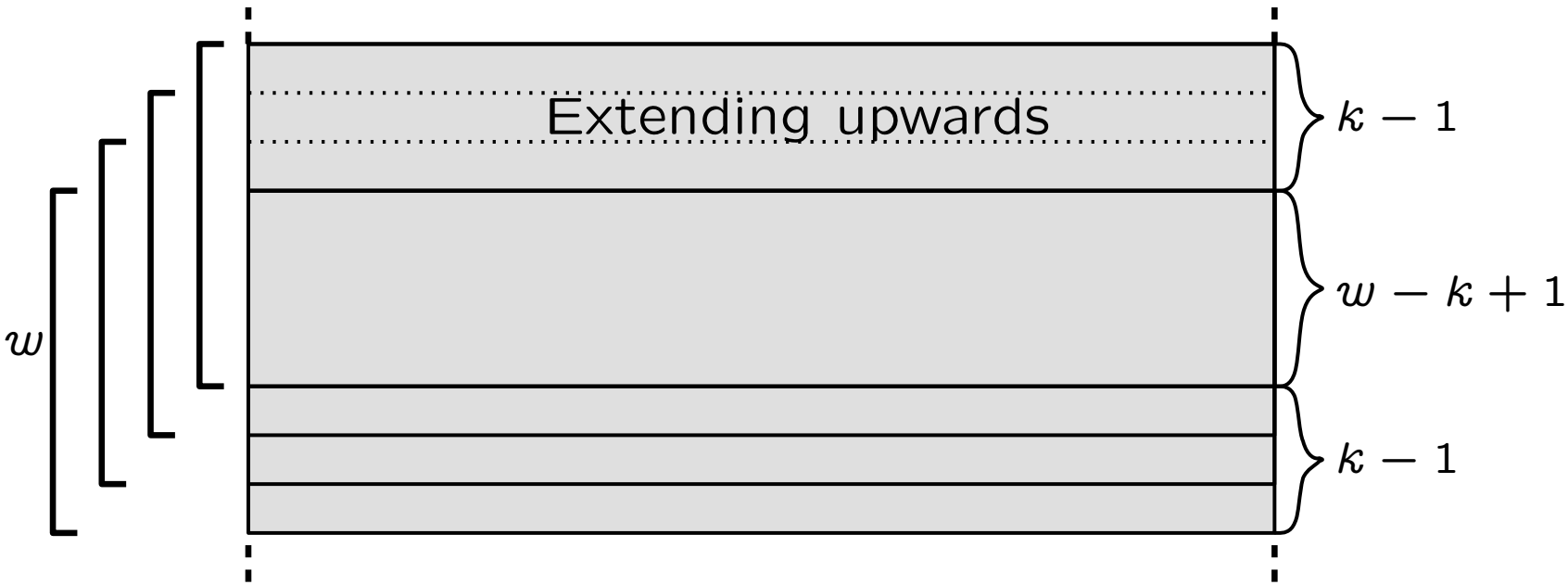
Computing Alignment Plots Using Seaweeds

Step 2: Work $c_2(n) = n \cdot (k-1)$



Computing Alignment Plots Using Seaweeds

Step 3: Work $c_3(n) = n \cdot \sum_{j=1}^{k-1} k - 1 - j$

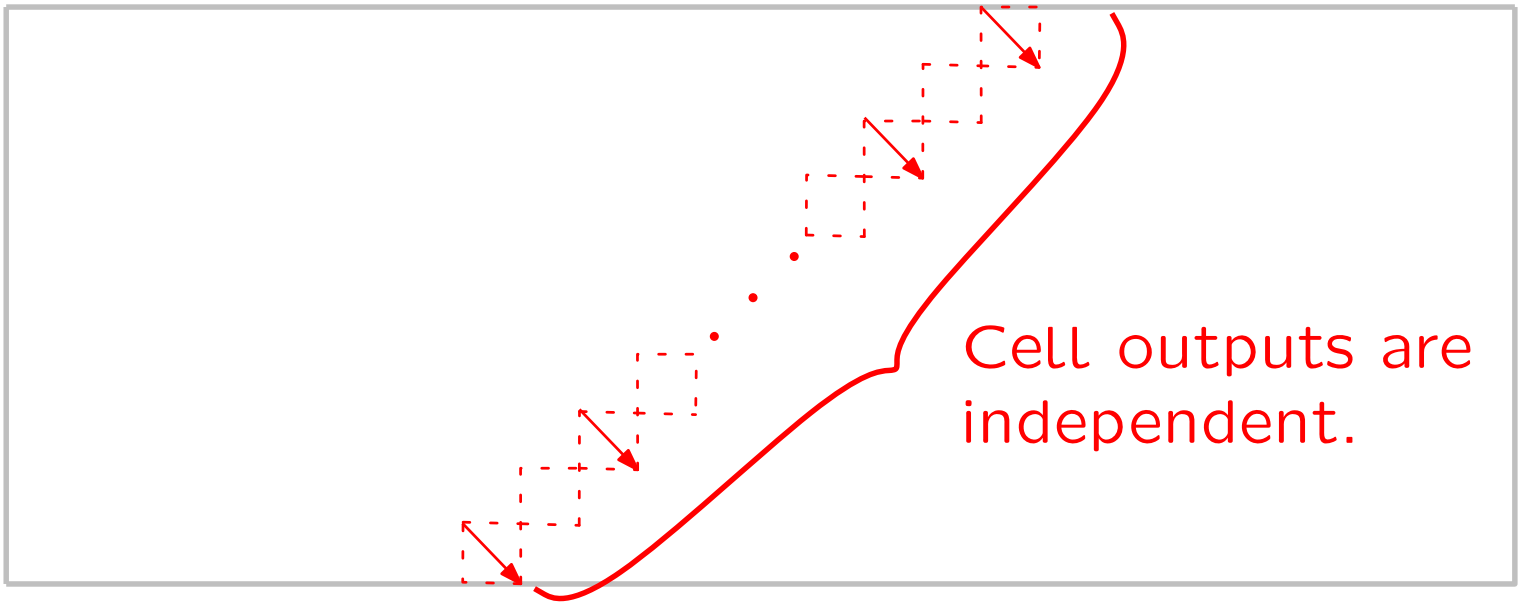


Computing Alignment Plots Using Seaweeds

Theoretically, optimal choice of k gives work $O(mn\sqrt{w})$ for computing the alignment plot.

Vector-Parallel Seaweeds

Standard solution: Process cells in a *wavefront* in parallel.



Outline

Implementation Notes

Current implementation uses C++ and Intel Assembly (x86 and x86_64, MMX/SSE2).

Explicit vectorisation of the inner loop using assembler code.

The core of the code consists of a small library for implementing operations on vectors of ω-bit integers.

Single CPU Execution Times

Data Set	**Mikey**	**Berti**	**Jimmy**	**Henry**
Input Size	2.7k × 0.6k	2.7k × 2.3k	15k × 97k	80k× 80k
Heur	5.1 (÷ 1.0)	41.1 (÷ 1.0)	2677 (÷ 1.0)	11708 (÷ 1.0)
BLCS	3.6 (÷ 1.4)	37.3 (÷ 1.1)	3680 (÷ 0.7)	16191 (÷ 0.7)
Sea-16	1.4 (÷ 3.6)	10.8 (÷ 3.8)	1026 (÷ 2.6)	4514 (÷ 2.6)
Sea-8	0.5 (÷ 10.2)	3.8 (÷ 10.8)	368 (÷ 7.3)	1614 (÷ 7.3)
Sea-8 SMP×2	0.3 (÷ 17.0)	3.4 (÷ 12.1)	210 (÷ 12.7)	821 (÷ 14.3)

(Execution times in seconds)

Vector-Parallelism: GPU vs. MMX/SSE

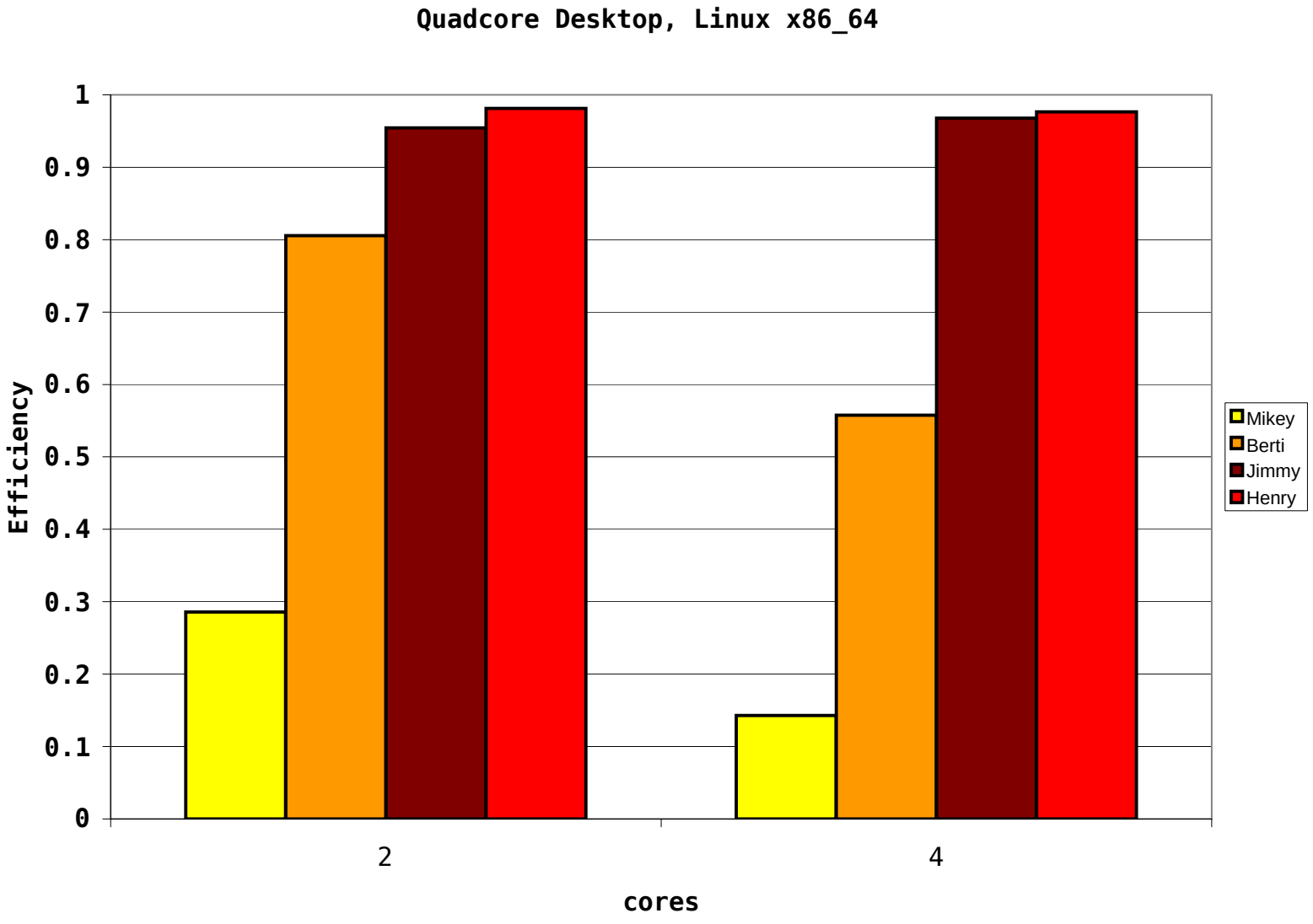
- Vector element size is important when using MMX/SSE: smaller vector elements allow higher degree of parallelism. We use 8 bits per seaweed for SSE.
- In the GPU implementation, we always work with 32 bits per seaweed.
- Therefore, the GPU version of the code can be used for larger window lengths.

Vector-Parallelism: GPU vs. MMX/SSE

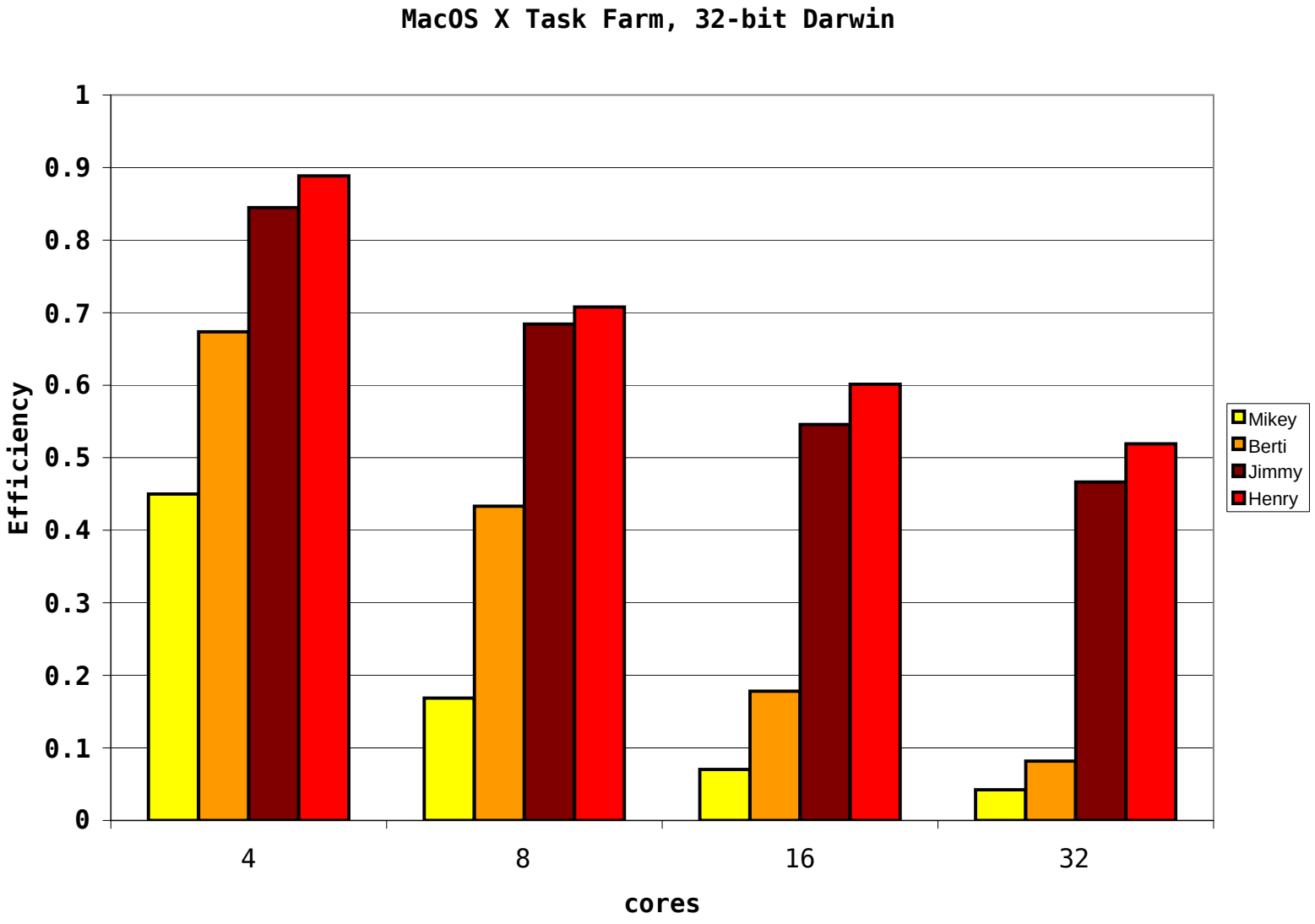
Data Set **Input Size**	**Berti** 2712×2305	**Jimmy** 15097×96901	**Henry** 80001×80001
Heur	40	2571	11708
Sea-nonoverlap-SSE2 ($k = 1$)	5.8	554	2410
Sea-nonoverlap-GPU ($k = 1$)	5.1	422	1759
Sea-overlap-GPU ($k = 4$)	4.8	381	1596

(Execution times in seconds)

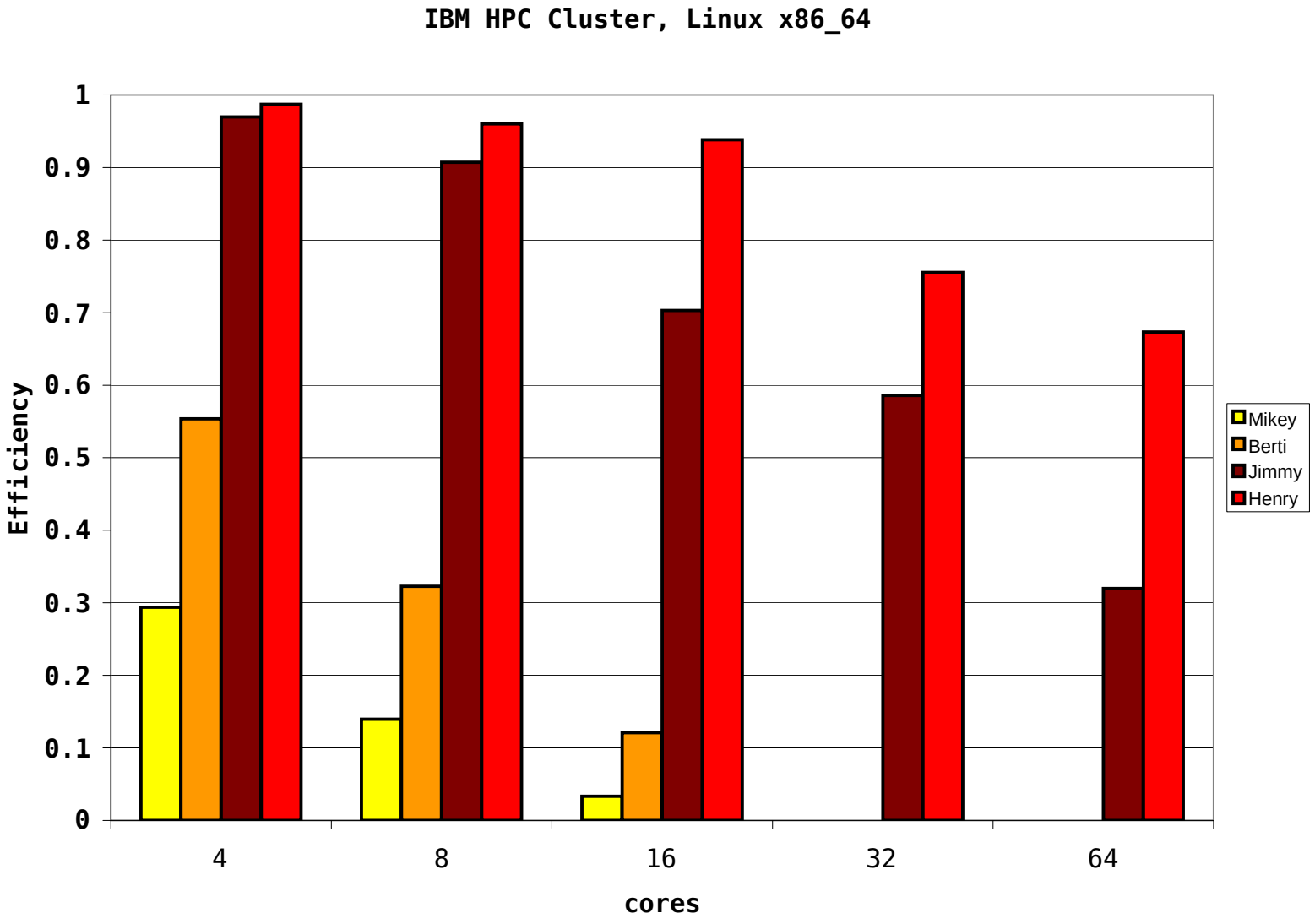
Parallel Efficiency using MPI



Parallel Efficiency using MPI



Parallel Efficiency using MPI



Outline

Summary

We have shown a new, fast algorithm for loss-free local sequence alignment.

Main contribution: reduced dependency of runtime on the length of the local alignments.

Method allows to use different types of parallelism.

Outlook

Better speedup for small problem sizes by smarter partitioning.

This is useful when using the code for small sequences as a web service, like BLAST.

Reduce implementation overhead for overlap method.

Less overhead for extending strips upwards allows better re-use of shared parts of the alignment dag.

Algorithmic improvements when handling more complex score matrices.

More complex score matrices make the search more sensitive.

Implement theoretically optimal $O(mn)$ method.

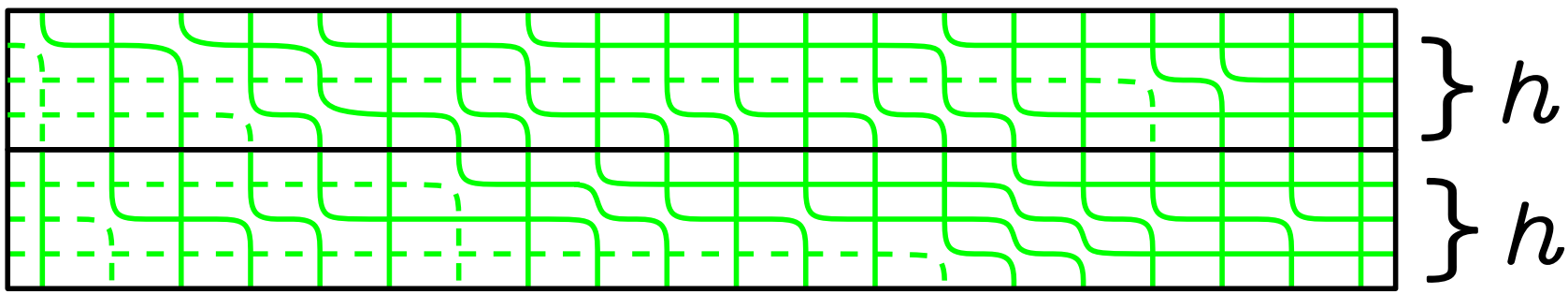
This uses distance multiplication, might not be practical.

Thanks for listening!

Questions?

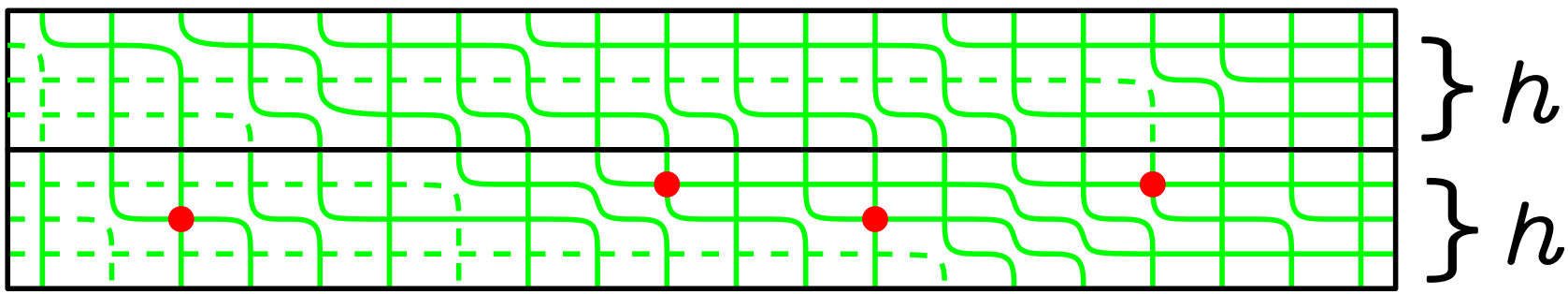
Window Size Independent Alignment Plots

Consider the seaweeds for two substrings of x sized h.



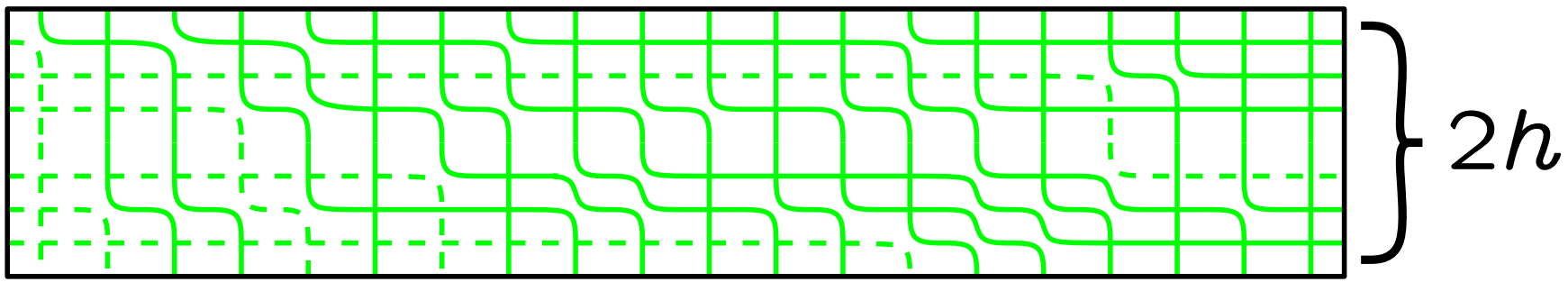
Window Size Independent Alignment Plots

We need to eliminate double-crossings to restore the seaweed behaviour.



Window Size Independent Alignment Plots

Then we obtain the seaweeds for a substring of x sized $2h$.



Window Size Independent Alignment Plots

Theorem (Tiskin:05)

Consider two substrings x_1 and x_2 of x with $|x_1| = |x_2| = h$.
Given the two implicit highest-score matrices for two strings x_1 and x_2 compared to y, we can obtain the implicit highest-score matrix for $x_1 x_2$ compared to y in time $O(h + n \log n)$.(*)

(*) x_1 and x_2 can have different lengths.

Window Size Independent Alignment Plots

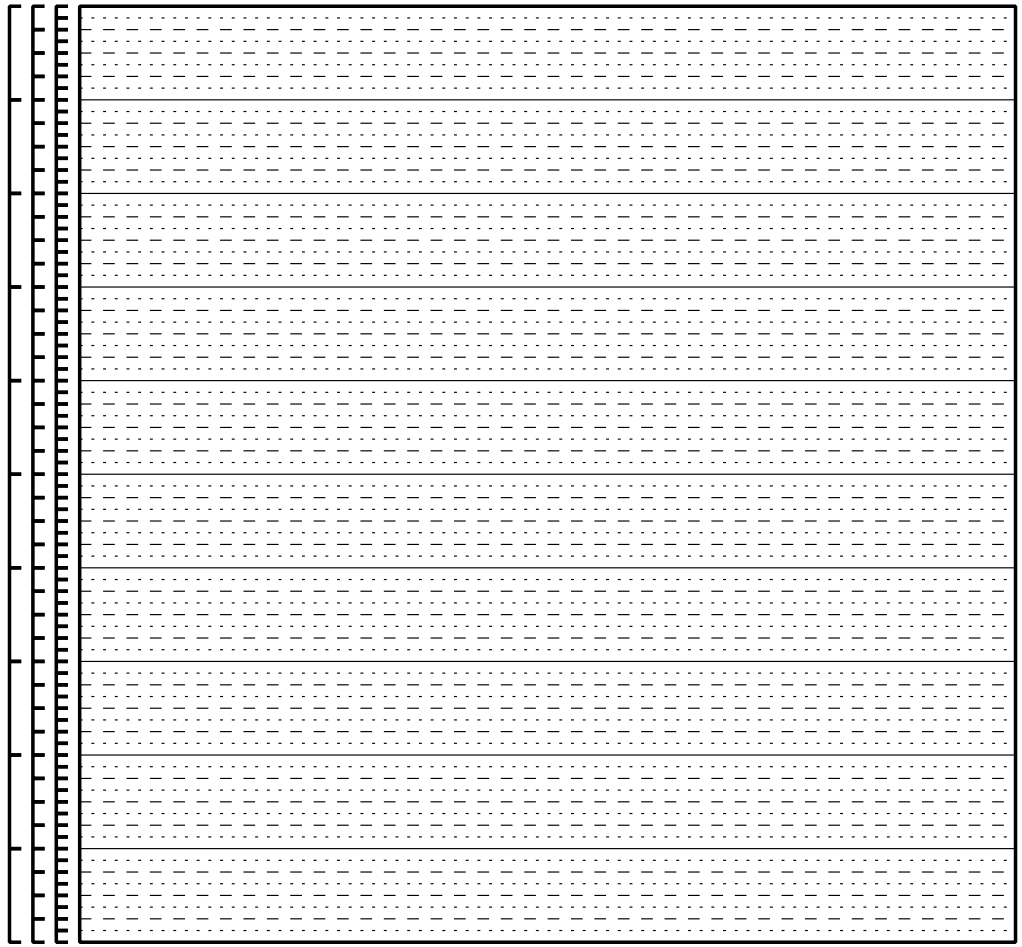
Theorem (Tiskin:05)

Consider two substrings x_1 and x_2 of x with $|x_1| = |x_2| = h$.
Given the two implicit highest-score matrices for two strings x_1 and x_2 compared to y, we can obtain the implicit highest-score matrix for $x_1 x_2$ compared to y in time $O(h + n \log n)$.(*)

(*) x_1 and x_2 can have different lengths.

Window Size Independent Alignment Plots

We pre-compute seaweeds for substrings of x sized $h \in \{1, 2, 4, \ldots, w\}$.



Window Size Independent Alignment Plots

We can obtain the seaweeds for every w-window in x by merging $O(\log w)$ pre-computed seaweed strips.