Periodic string comparison

Alexander Tiskin

Department of Computer Science
University of Warwick
http://www.dcs.warwick.ac.uk/~tiskin

Introduction

String matching: finding an exact pattern in a string

String comparison: finding similar patterns in two strings

(Also known as "approximate string matching", no relation to approximation algorithms!)

Applications: computational biology, image recognition, ...

Introduction

String matching: finding an exact pattern in a string

String comparison: finding similar patterns in two strings

(Also known as "approximate string matching", no relation to approximation algorithms!)

Applications: computational biology, image recognition, ...

Standard types of string comparison:

- global: whole string vs whole string
- local: substrings vs substrings

Main focus of this work:

- semi-local: whole string vs substrings; prefixes vs suffixes

Main tool: implicit unit-Monge matrices

Integers: $\ldots -2, -1, 0, 1, 2, \ldots$

Odd half-integers: $\ldots -\frac{5}{2}, -\frac{3}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \ldots$

We consider finite and infinite integer matrices over integer and odd half-integer indices. For simplicity, index range will usually be ignored.

A permutation matrix is a $0/1$ matrix with exactly one nonzero per row and per column

Given matrix D, its distribution matrix is $D^{\Sigma}(i,j) = \sum_{i'>i, j'<j} D(i',j')$

In other words, $D^{\Sigma}(i,j)$ is the sum of all $D(i',j')$, where (i',j') is dominated by (i,j)

Given matrix D, its distribution matrix is $D^{\Sigma}(i,j) = \sum_{i'>i,j'<j} D(i',j')$

In other words, $D^{\Sigma}(i,j)$ is the sum of all $D(i',j')$, where (i',j') is dominated by (i,j)

Given matrix E, its density matrix is

$$E^{\square}(i,j) = E(i^-,j^+) - E(i^-,j^-) - E(i^+,j^+) + E(i^+,j^-)$$

where $i^{\pm} = i \pm \frac{1}{2}$; D^{Σ}, E over integers; D, $E^{\square}$ over odd half-integers

Given matrix D, its distribution matrix is $D^{\Sigma}(i,j) = \sum_{i'>i,j'<j} D(i',j')$

In other words, $D^{\Sigma}(i,j)$ is the sum of all $D(i',j')$, where (i',j') is dominated by (i,j)

Given matrix E, its density matrix is

$$E^{\square}(i,j) = E(i^-,j^+) - E(i^-,j^-) - E(i^+,j^+) + E(i^+,j^-)$$

where $i^{\pm} = i \pm \frac{1}{2}$; D^{Σ}, E over integers; D, $E^{\square}$ over odd half-integers

$(D^{\Sigma})^{\square} = D$ for all D

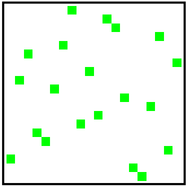
Matrix E is simple, if $(E^{\square})^{\Sigma} = E$

Matrix E is Monge, if $E^{\square}$ is nonnegative

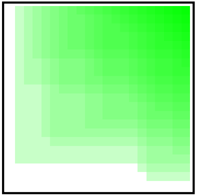
Intuition: border-to-border distances in a (weighted) planar graph

Matrix E is unit-Monge, if $E^{\square}$ is a permutation matrix

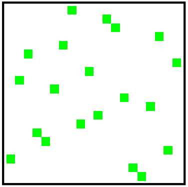
Intuition: border-to-border distances in a grid-like graph



P



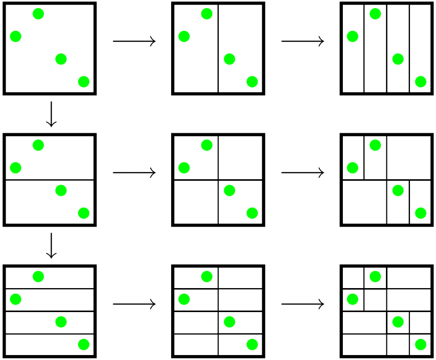
P^{Σ}



$(P^{\Sigma})^{\square} = P$

Implicit P^{Σ}: range tree on nonzeros of P [Bentley: 1980]

- binary search tree by i-coordinate
- under every node, binary search tree by j-coordinate



Implicit P^{Σ} (contd.)

Every node of the range tree represents a canonical range (rectangular region), and stores its nonzero count

Overall, $\leq n \log n$ canonical ranges are non-empty

Range tree supports dominance counting queries: how many nonzeros are dominated by a given point? Answered by decomposing query range into $\leq \log^2 n$ disjoint canonical ranges.

Total size $O(n \log n)$, query time $O(\log^2 n)$

There are asymptotically more efficient (but less practical) data structures

Matrix $\odot$-multiplication (a.k.a. distance, $(\min, +)$ or tropical multiplication)

$A \odot B = C \qquad C(i,k) = \min_j\bigl(A(i,j) + B(j,k)\bigr)$

Matrix $\odot$-multiplication (a.k.a. distance, $(\min, +)$ or tropical multiplication)

$A \odot B = C \qquad C(i,k) = \min_j \bigl(A(i,j) + B(j,k)\bigr)$

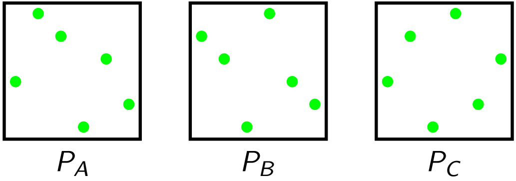
Matrix classes closed under $\odot$-multiplication:

- general numerical (integer, real) matrices
- Monge matrices
- simple unit-Monge matrices

Simple unit-Monge matrices of size n form an aperiodic monoid (i.e. a monoid as far as possible from a group) under $\odot$-multiplication

We call it the seaweed monoid $\mathcal{T}_n$

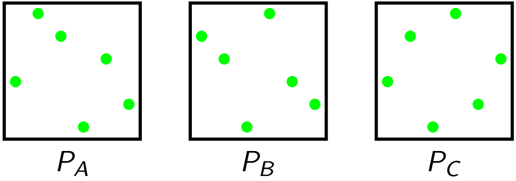
Simple unit-Monge matrices: $\odot$-multiplication = seaweed composition

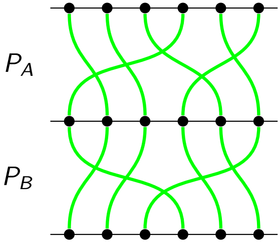


$$P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$$

Simple unit-Monge matrices: $\odot$-multiplication = seaweed composition

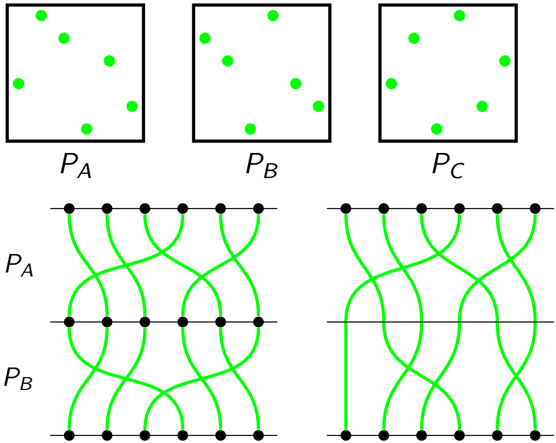
$$P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$$





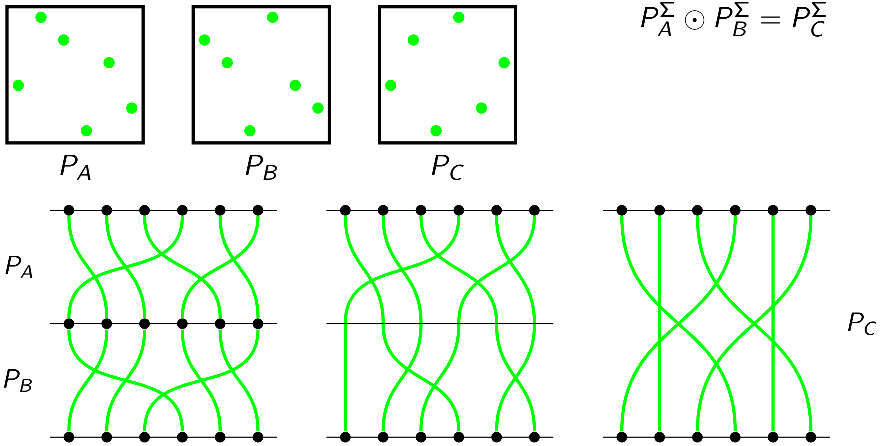
Simple unit-Monge matrices: $\odot$-multiplication = seaweed composition

$$P_A^{\Sigma} \odot P_B^{\Sigma} = P_C^{\Sigma}$$



Simple unit-Monge matrices: $\odot$-multiplication = seaweed composition

$$P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$$



Seaweeds: similar to braids, generated by strand crossings

Unlike in braids, all seaweed crossings are

- level (not underpass/overpass)
- idempotent, i.e. two seaweeds can cross at most once

Seaweed composition: associative, no inverse (a crossing cannot be cancelled)

Identity: $1 \odot x = x$, no seaweeds crossing

Zero: $0 \odot x = 0$, all seaweeds crossing

$$1 = \begin{pmatrix} \bullet & \cdot & \cdot & \cdot & \cdot \\ \cdot & \bullet & \cdot & \cdot & \cdot \\ \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \cdot & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \cdot & \bullet \end{pmatrix}^{\Sigma} \qquad 0 = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \bullet \\ \cdot & \cdot & \cdot & \bullet & \cdot \\ \cdot & \cdot & \bullet & \cdot & \cdot \\ \cdot & \bullet & \cdot & \cdot & \cdot \\ \bullet & \cdot & \cdot & \cdot & \cdot \end{pmatrix}^{\Sigma}$$

The seaweed monoid $\mathcal{T}_n$:

- $n!$ elements (permutations of size n)
- $n-1$ generators $g_1, g_2, \ldots, g_{n-1}$ (elementary crossings)

$g_i^2 = g_i$ for all i (idempotence)

$g_i g_j = g_j g_i$ $j - i > 1$ (far commutativity)

$g_i g_j g_i = g_j g_i g_j$ $j - i = 1$ (braid relations)

Computation: a confluent rewriting system can be obtained by software (Semigroupe, GAP)

Generalisation: Coxeter monoids (subgroup monoids in groups)

[Tsaranov: 90]

The seaweed monoid $\mathcal{T}_3$

Generators: 1, $a = g_1$, $b = g_2$

Other elements: ab, ba, $aba = 0$

Rewriting system:

$aa \longrightarrow a$
$bb \longrightarrow b$
$bab \longrightarrow 0$
$aba \longrightarrow 0$

The seaweed monoid $\mathcal{T}_4$

Generators: 1, $a = g_1$, $b = g_2$, $c = g_3$

Other elements: ab, ac, ba, bc, cb, aba, abc, acb, bac, bcb, cba, $abac$, $abcb$, $acba$, $bacb$, $bcba$, $abacb$, $abcba$, $bacba$, $abacba = 0$

Rewriting system:

$aa \longrightarrow a$
$bb \longrightarrow b$
$ca \longrightarrow ac$
$cc \longrightarrow c$
$bab \longrightarrow aba$
$cbc \longrightarrow bcb$
$cbac \longrightarrow bcba$
$abacba \longrightarrow 0$

The implicit matrix $\odot$-multiplication problem

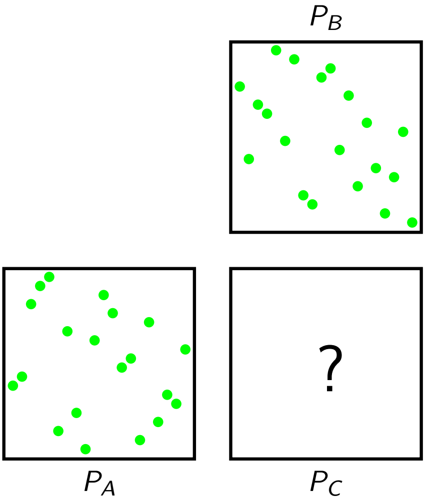
Given permutation matrices P_A, P_B, compute P_C, such that $P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$

The implicit matrix $\odot$-multiplication problem

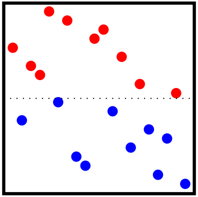
Given permutation matrices P_A, P_B, compute P_C, such that $P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$

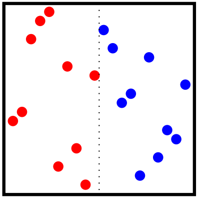
Matrix $\odot$-multiplication: running time

matrix type	time	
general	$O(n^3)$	standard
Monge	$O(n^2)$	by [Aggarwal+: 1987]
implicit simple unit-Monge (P^Σ)		
	$O(n^{1.5})$	[T: 2006]
	$O(n \log n)$	[T: NEW]


P_B
?
P_A
P_C

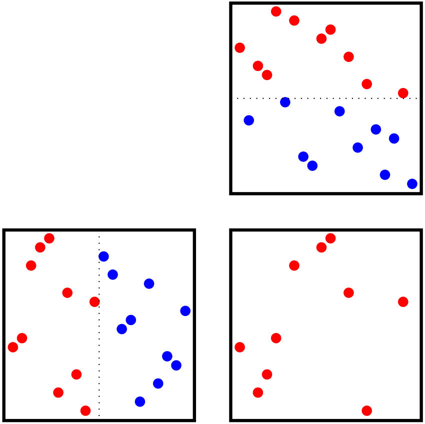
$P_{B,lo},\ P_{B,hi}$



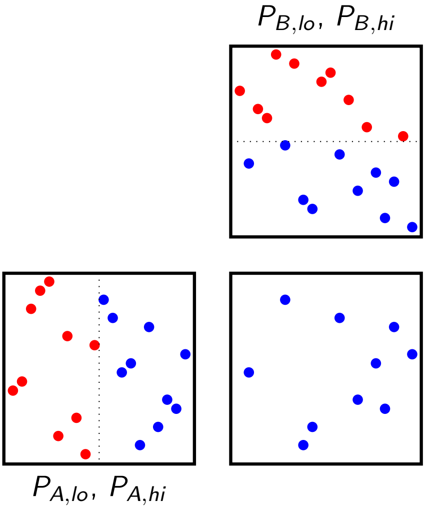


$P_{A,lo},\ P_{A,hi}$

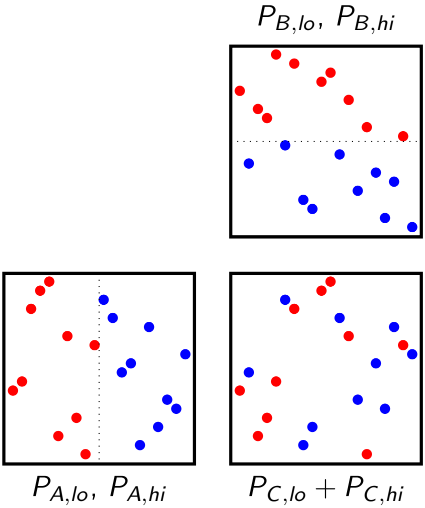
$P_{B,lo}$, $P_{B,hi}$



$P_{A,lo}$, $P_{A,hi}$



$P_{B,lo}$, $P_{B,hi}$
$P_{A,lo}$, $P_{A,hi}$


$P_{B,lo}$, $P_{B,hi}$
$P_{A,lo}$, $P_{A,hi}$
$P_{C,lo} + P_{C,hi}$

Implicit matrix $\odot$-multiplication: the algorithm

$P^{\Sigma}_{C}(i,k) = \min_j \left(P^{\Sigma}_{A}(i,j) + P^{\Sigma}_{B}(j,k)\right)$

Divide-and-conquer on the range of j

Divide P_A horizontally, P_B vertically; two subproblems of effective size $n/2$:

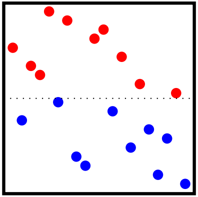
$P^{\Sigma}_{A,lo} \odot P^{\Sigma}_{B,lo} = P^{\Sigma}_{C,lo} \qquad P^{\Sigma}_{A,hi} \odot P^{\Sigma}_{B,hi} = P^{\Sigma}_{C,hi}$

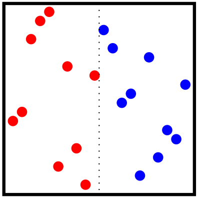
Conquer: most (but not all!) nonzeros of $P_{C,lo}$, $P_{C,hi}$ appear in P_C

Missing nonzeros can be obtained in time $O(n)$ using the Monge property

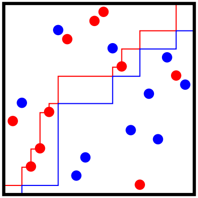
Overall time $O(n \log n)$

$P_{B,lo}, P_{B,hi}$

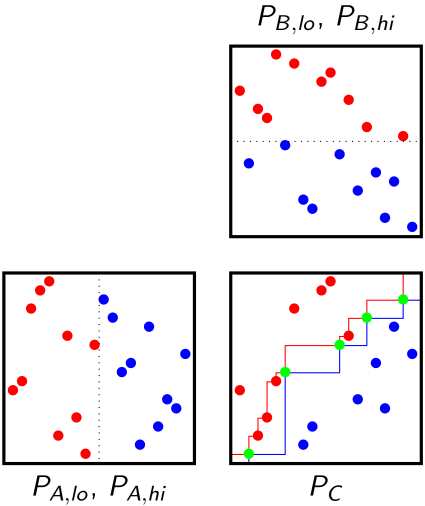




$P_{A,lo}, P_{A,hi}$



$P_{C,lo} + P_{C,hi}$



$P_{B,lo}, P_{B,hi}$
$P_{A,lo}, P_{A,hi}$
P_C

Consider strings (= sequences) over an alphabet of size σ

Distinguish contiguous substrings and not necessarily contiguous subsequences

Special cases of substring: prefix, suffix

Notation: strings a, b of length m, n respectively

Assume where necessary: $m \leq n$; m, n reasonably close

Consider strings (= sequences) over an alphabet of size σ

Distinguish contiguous substrings and not necessarily contiguous subsequences

Special cases of substring: prefix, suffix

Notation: strings a, b of length m, n respectively

Assume where necessary: $m \leq n$; m, n reasonably close

The longest common subsequence (LCS) score:

- length of longest string that is a subsequence of both a and b
- equivalently, alignment score, where $score(match) = 1$ and $score(mismatch) = 0$

The LCS problem

Give the LCS score for a vs b

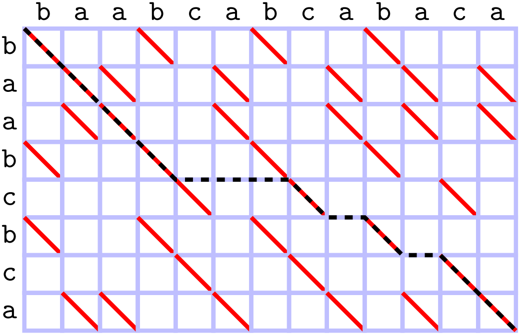
The LCS problem

Give the LCS score for a vs b

LCS: running time

$O(mn)$		[Wagner, Fischer: 1974]
$O\left(\frac{mn}{\log n}\right)$	$\sigma = O(1)$	[Masek, Paterson: 1980]
		[Crochemore+: 2003]
$O\left(\frac{mn(\log \log n)^2}{\log n}\right)$		[Paterson, Dancik: 1994]

LCS on the alignment graph (directed, acyclic)



blue = 0
red = 1

LCS("baabcbca", "baabcabcabaca") = "baabcbca"

LCS = highest-score corner-to-corner path

LCS: dynamic programming (DP) algorithm [Wagner, Fischer: 1974]

Sweep alignment graph, respecting node dependencies

Running time $O(mn)$

LCS: dynamic programming (DP) algorithm [Wagner, Fischer: 1974]

Sweep alignment graph, respecting node dependencies

Running time $O(mn)$

LCS: micro-block DP algorithm [Masek, Paterson: 1980]

Sweep alignment graph in square blocks, respecting block dependencies

Block size: $t = O(\log n)$

Block interface: $O(t)$ inputs/outputs, each of size $O(\log \sigma)$

Use precomputed mapping of all possible input/output combinations

Running time $O\left(\frac{mn}{\log n}\right)$ when $\sigma = O(1)$, even on log-cost RAM

The semi-local LCS problem

Give the (implicit) matrix of $O(m^2 + n^2)$ LCS scores:

- string-substring LCS: string a vs every substring of b
- prefix-suffix LCS: every prefix of a vs every suffix of b
- symmetrically, substring-string and suffix-prefix LCS

The semi-local LCS problem

Give the (implicit) matrix of $O(m^2 + n^2)$ LCS scores:

- string-substring LCS: string a vs every substring of b
- prefix-suffix LCS: every prefix of a vs every suffix of b
- symmetrically, substring-string and suffix-prefix LCS

The three-way semi-local LCS problem

Give the (implicit) matrix of $O(n^2)$ LCS scores:

- string-substring, prefix-suffix, suffix-prefix LCS
- no substring-string LCS

Suitable for $m \gg n$

The semi-local LCS problem

Give the (implicit) matrix of $O(m^2 + n^2)$ LCS scores:

- string-substring LCS: string a vs every substring of b
- prefix-suffix LCS: every prefix of a vs every suffix of b
- symmetrically, substring-string and suffix-prefix LCS

The three-way semi-local LCS problem

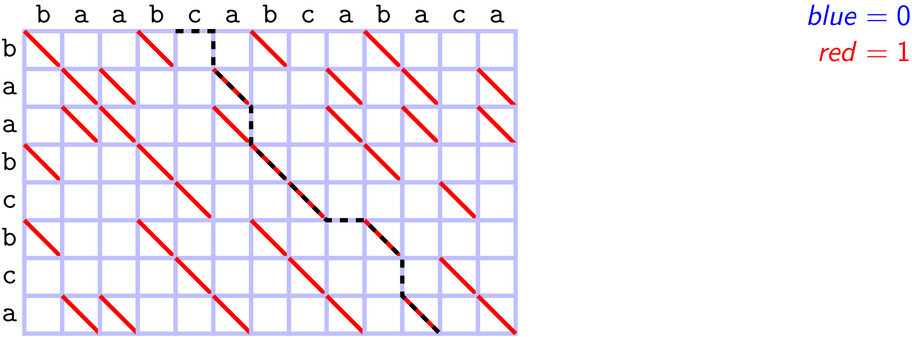
Give the (implicit) matrix of $O(n^2)$ LCS scores:

- string-substring, prefix-suffix, suffix-prefix LCS
- no substring-string LCS

Suitable for $m \gg n$

Cf.: dynamic programming gives prefix-prefix LCS

Semi-local LCS on the alignment graph



$LCS(\texttt{"baabcbca"}, \texttt{"...cabcaba..."}) = \texttt{"abcba"}$

Semi-local LCS = all highest-score border-to-border paths
(string-substring = top-to-bottom, etc.)

The LCS problem is a special case of the alignment score problem with weighted matches, mismatches and gaps

- LCS score: $w_{match} = 1$, $w_{mismatch} = w_{gap} = 0$
- Levenshtein score: $w_{match} = 2$, $w_{mismatch} = 1$, $w_{gap} = 0$

An alignment score is rational, if w_{match}, $w_{mismatch}$, w_{gap} are rational; reduces to LCS score by constant-factor blow-up of alignment graph

The LCS problem is a special case of the alignment score problem with weighted matches, mismatches and gaps

- LCS score: $w_{match} = 1$, $w_{mismatch} = w_{gap} = 0$
- Levenshtein score: $w_{match} = 2$, $w_{mismatch} = 1$, $w_{gap} = 0$

An alignment score is rational, if w_{match}, $w_{mismatch}$, w_{gap} are rational; reduces to LCS score by constant-factor blow-up of alignment graph

The semi-local alignment score problem: string-substring, prefix-suffix, substring-string, suffix-prefix alignment scores

Edit distance: minimum cost to transform a into b by weighted character edits (insertion, deletion, substitution)

The semi-local edit distance problem: semi-local alignment score problem with $w_{match} = 0$, $w_{mismatch} = -w_{sub}$, $w_{gap} = -w_{indel}$

The semi-local LCS score matrix

0	1	2	3	4	5	6	6	7	8	8	8	8	8
−1	0	1	2	3	4	5	5	6	7	7	7	7	7
−2	−1	0	1	2	3	4	4	5	6	6	6	6	7
−3	−2	−1	0	1	2	3	3	4	5	5	6	6	7
−4	−3	−2	−1	0	1	2	2	3	4	4	(5)	5	6
−5	−4	−3	−2	−1	0	1	2	3	4	4	5	5	6
−6	−5	−4	−3	−2	−1	0	1	2	3	3	4	4	5
−7	−6	−5	−4	−3	−2	−1	0	1	2	2	3	3	4
−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	3	4
−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4
−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3
−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2
−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1
−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0

$a =$ "baabcbca"

$b =$ "baabcabcabaca"

$A(0, 13) = LCS(a, b) = 8$

$b' =$ "...cabcaba..."

$A(4, 11) = LCS(a, b') = 5$

$A(i, j) = j - i$ if $i > j$

Semi-local LCS: output representation and running time

size	query time		
$O(n^2)$	$O(1)$		trivial
$O(m^{1/2}n)$	$O(\log n)$	string-substring	[Alves+: 2003]
$O(n)$	$O(n)$	string-substring	[Alves+: 2005]
$O(n \log n)$	$O(\log^2 n)$		[T: 2006]

running time		
$O(mn^2)$		naive
$O(mn)$	string-substring	[Schmidt: 1998]
	string-substring	[Alves+: 2005]
$O(mn)$		[T: 2006]
$O\left(\frac{mn}{\log^{0.5} n}\right)$		[T: 2006]
$O\left(\frac{mn(\log \log n)^2}{\log n}\right)$		[T: 2007]

A: the semi-local LCS score matrix for a vs b

$A(i, j)$: the number of matched characters for a vs substring of b

$Q(i, j) = j - i - A(i, j)$: the number of unmatched characters

Properties of matrix Q:

- Q is simple unit-Monge
- therefore, $Q = P^{\Sigma}$ for some permutation matrix P

$P = Q^{\square} = -A^{\square}$ is an implicit representation of A

Range tree for P: memory $O(n \log n)$, query time $O(\log^2 n)$

The semi-local LCS score matrix

0	1	2	3	4	5	6	6	7	8	8	8	8	8
−1	0	1	2	3	4	5	5	6	7	7	7	7	7
−2	−1	0	1	2	3	4	4	5	6	6	6	6	7
−3	−2	−1	0	1	2	3	3	4	5	5	6	6	7
−4	−3	−2	−1	0	1	2	2	3	4	4	5	5	6
−5	−4	−3	−2	−1	0	1	2	3	4	4	5	5	6
−6	−5	−4	−3	−2	−1	0	1	2	3	3	4	4	5
−7	−6	−5	−4	−3	−2	−1	0	1	2	2	3	3	4
−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	3	4
−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4
−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3
−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2
−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1
−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0

$a =$ "baabcbca"

$b =$ "baabcabcabaca"

$b' =$ "...cabcaba..."

$A(4, 11) = LCS(a, b') = 5$

$A(i, j) = j - i$ if $i > j$

The semi-local LCS score matrix

0	1	2	3	4	5	6	6	7	8	8	8	8	8
−1	0	1	2	3	4	5	5	6	7	7	7	7	7
−2	−1	0	1	2	3	4	4	5	6	6	6	6	7
−3	−2	−1	0	1	2	3	3	4	5	5	6	6	7
−4	−3	−2	−1	0	1	2	2	3	4	4	(5)	5	6
−5	−4	−3	−2	−1	0	1	2	3	4	4	5	5	6
−6	−5	−4	−3	−2	−1	0	1	2	3	3	4	4	5
−7	−6	−5	−4	−3	−2	−1	0	1	2	2	3	3	4
−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	3	4
−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4
−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3
−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2
−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1
−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0

$a =$ "baabcbca"

$b =$ "baabcabcabaca"

$b' =$ "...cabcaba..."

$A(4, 11) = LCS(a, b') = 5$

$A(i, j) = j - i$ if $i > j$

blue: difference 0

red: difference 1

Semi-local string comparison

Highest-score matrices

The semi-local LCS score matrix

0	1	2	3	4	5	6	6	7	8	8	8	8	8
−1	0	1	2	3	4	5	5	6	7	7	7	7	7
−2	−1	0	1	2	3	4	4	5	6	6	6	6	7
−3	−2	−1	0	1	2	3	3	4	5	5	6	6	7
−4	−3	−2	−1	0	1	2	2	3	4	4	(5)	5	6
−5	−4	−3	−2	−1	0	1	2	3	4	4	5	5	6
−6	−5	−4	−3	−2	−1	0	1	2	3	3	4	4	5
−7	−6	−5	−4	−3	−2	−1	0	1	2	2	3	3	4
−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	3	4
−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3	4
−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2	3
−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1	2
−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0	1
−13	−12	−11	−10	−9	−8	−7	−6	−5	−4	−3	−2	−1	0

$a =$ `"baabcbca"`

$b =$ `"baabcabcabaca"`

$b' =$ `"...cabcaba..."`

$A(4, 11) = LCS(a, b') = 5$

$A(i, j) = j - i$ if $i > j$

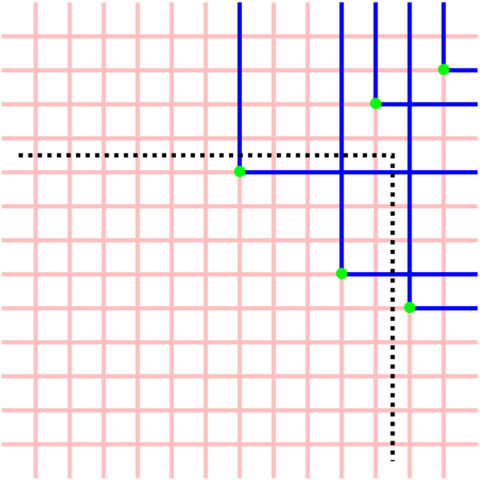
blue: difference 0

red: difference 1

green: $P(i, j) = 1$

$A(i, j) = j - i - P^{\Sigma}(i, j)$

The semi-local LCS score matrix



$a =$ "baabcbca"

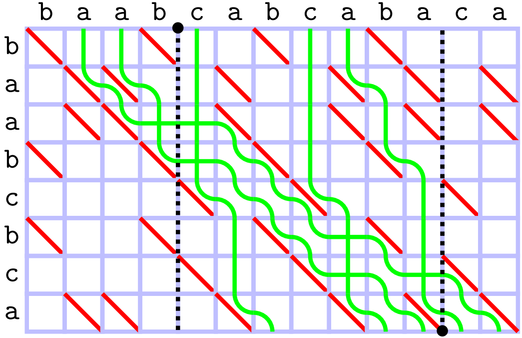
$b =$ "baabcabcabaca"

$b' =$ "...cabcaba..."

$A(4, 11) = LCS(a, b') =$
$11 - 4 - P^{\Sigma}(i, j) =$
$11 - 4 - 2 = 5$

P gives an implicit representation of A

The seaweeds in the alignment graph



$a =$ "baabcbca"

$b =$ "baabcabcabaca"

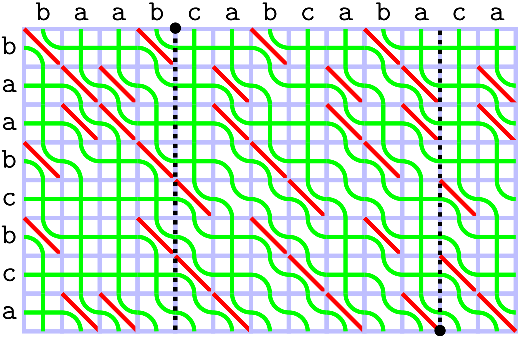
$b' =$ "...cabcaba..."

$A(4, 11) = LCS(a, b') =$
$11 - 4 - P^{\Sigma}(i, j) =$
$11 - 4 - 2 = 5$

P gives an implicit representation of A

$P(i, j) = 1$ corresponds to seaweed $(top, i) \rightsquigarrow (bottom, j)$

The seaweeds in the alignment graph



$a =$ "baabcbca"

$b =$ "baabcabcabaca"

$b' =$ "...cabcaba..."

$A(4, 11) = LCS(a, b') =$
$11 - 4 - P^{\Sigma}(i, j) =$
$11 - 4 - 2 = 5$

P gives an implicit representation of A

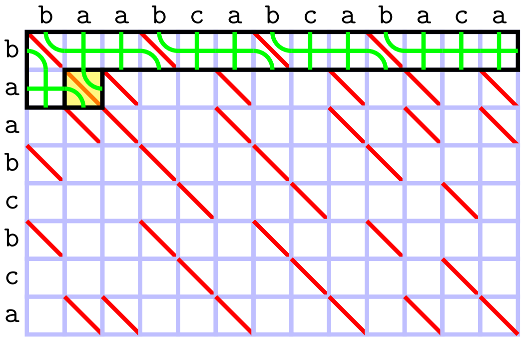
$P(i, j) = 1$ corresponds to seaweed $(top, i) \rightsquigarrow (bottom, j)$

Also define $top \rightsquigarrow right$, $left \rightsquigarrow right$, $left \rightsquigarrow bottom$ seaweeds

Gives complete border-to-border graph-theoretic matching

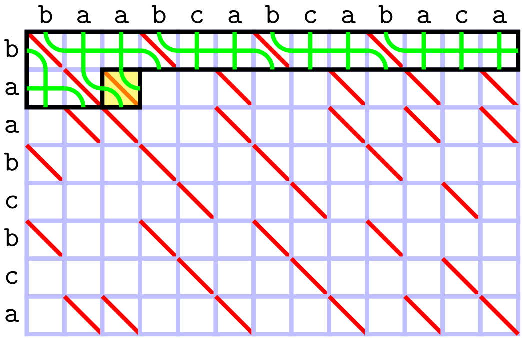
The seaweed algorithm

The seaweed algorithm



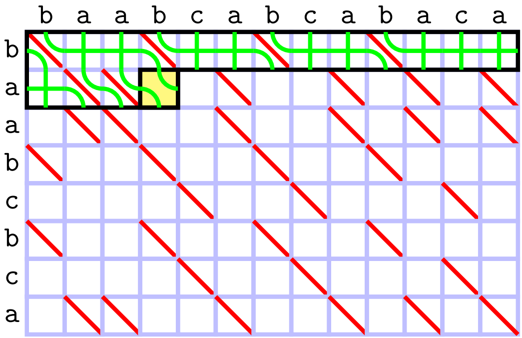
The seaweed algorithm

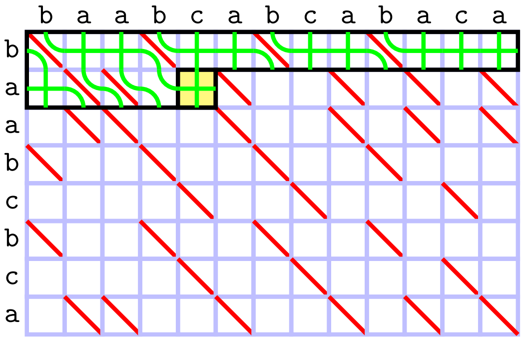
The seaweed algorithm



The seaweed algorithm

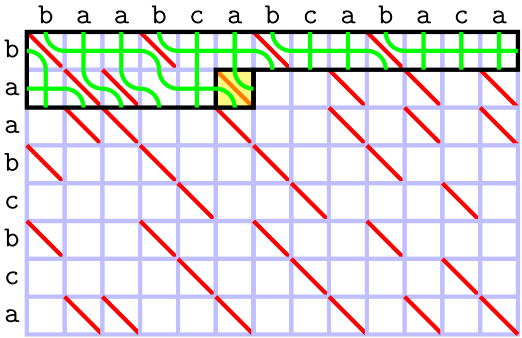
The seaweed algorithm




b a a b c a b c a b a c a
b
a
a
b
c
b
c
a

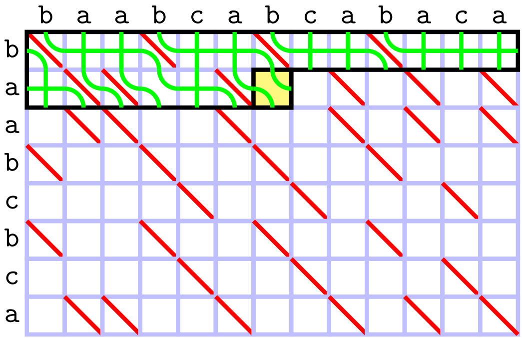
The seaweed algorithm

The seaweed algorithm



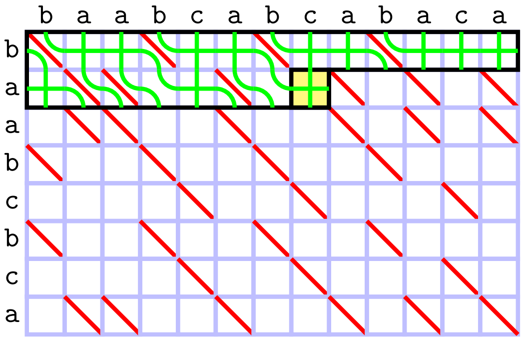
The seaweed algorithm

The seaweed algorithm



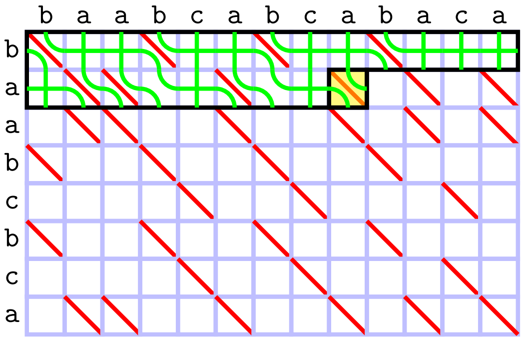
The seaweed algorithm

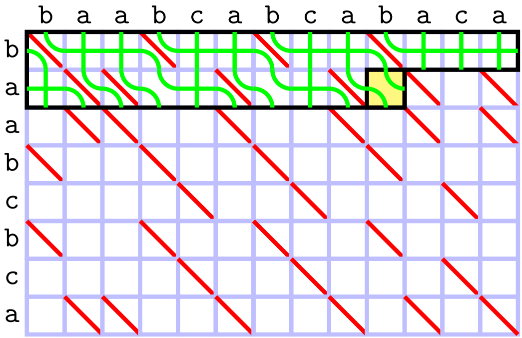
The seaweed algorithm



The seaweed algorithm

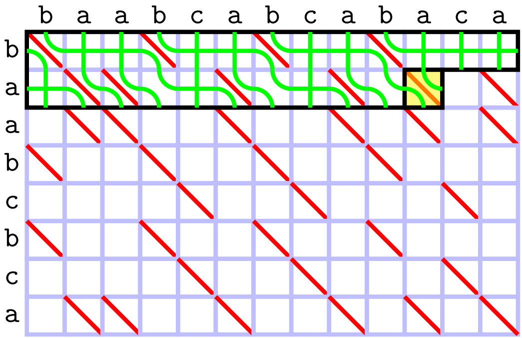
The seaweed algorithm




b a a b c a b c a b a c a
b
a
a
b
c
b
c
a

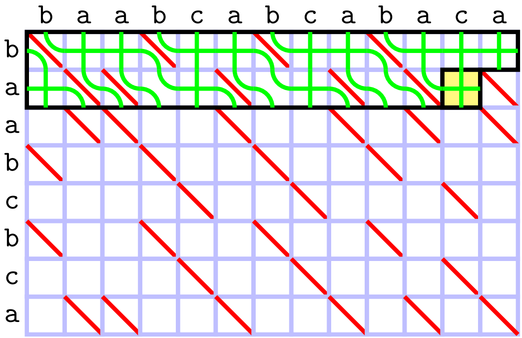
The seaweed algorithm

The seaweed algorithm



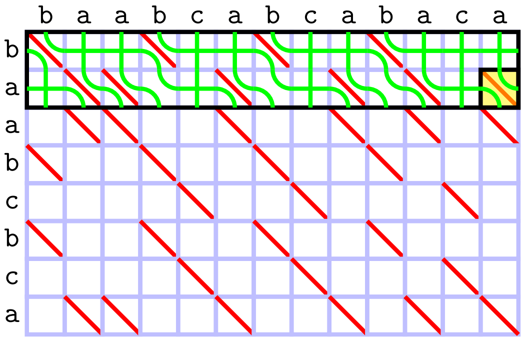
The seaweed algorithm

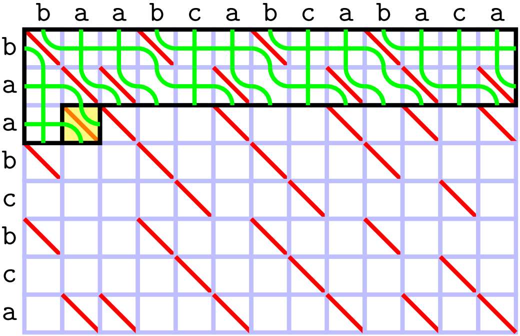
The seaweed algorithm



The seaweed algorithm

The seaweed algorithm

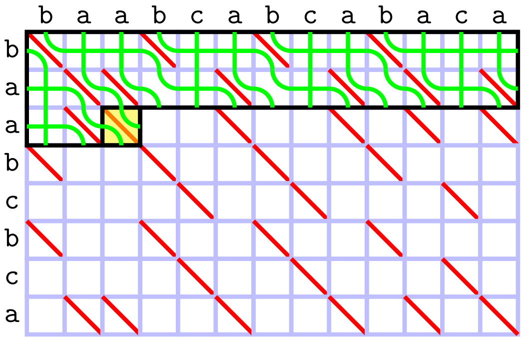




b a a b c a b c a b a c a
b
a
a
b
c
b
c
a

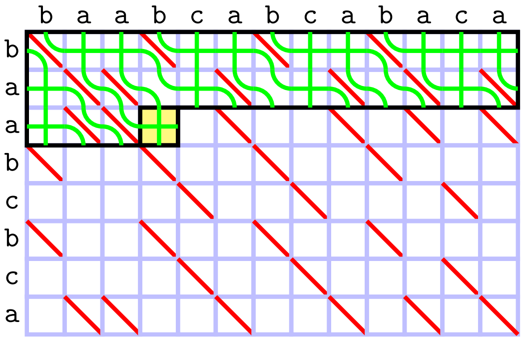
The seaweed algorithm

The seaweed algorithm



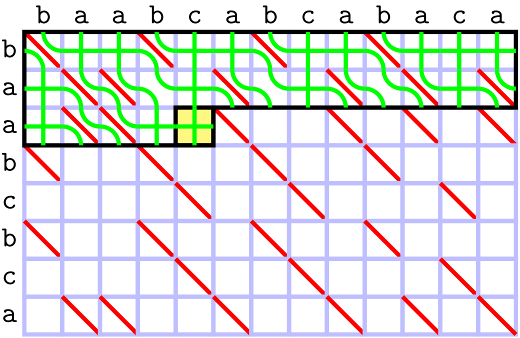
The seaweed algorithm

The seaweed algorithm



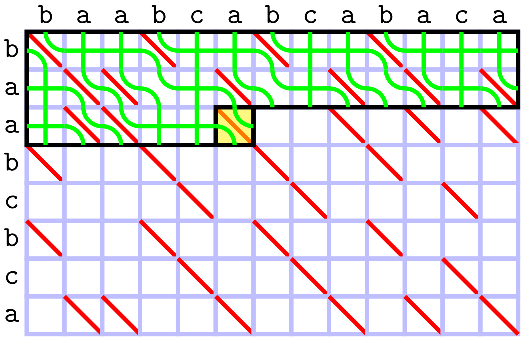
The seaweed algorithm

The seaweed algorithm



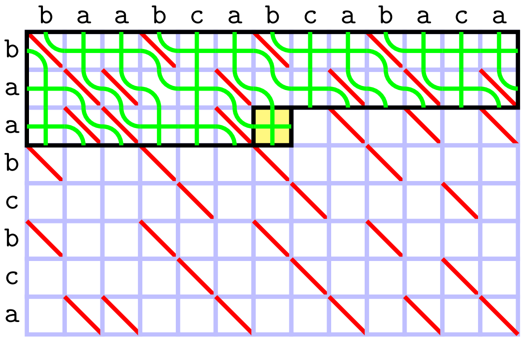
The seaweed algorithm

The seaweed algorithm



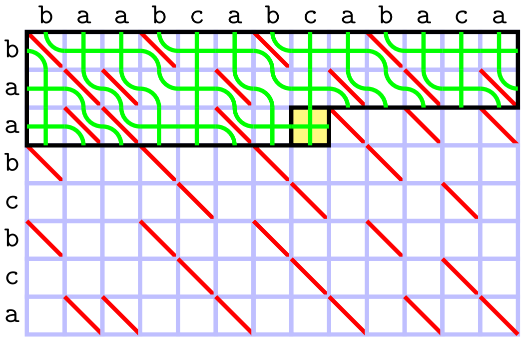
The seaweed algorithm

The seaweed algorithm



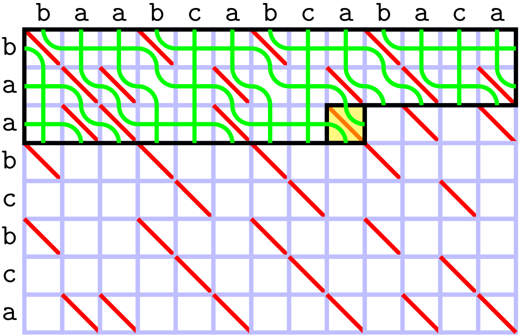
The seaweed algorithm

The seaweed algorithm



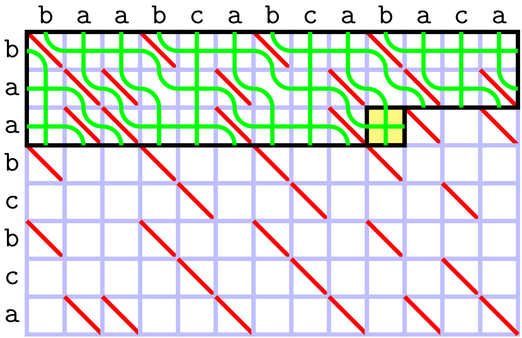
The seaweed algorithm

The seaweed algorithm



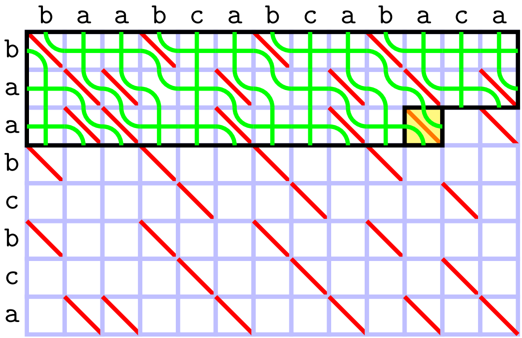
The seaweed algorithm

The seaweed algorithm



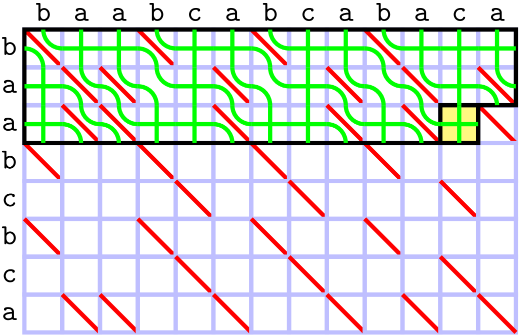
The seaweed algorithm

The seaweed algorithm



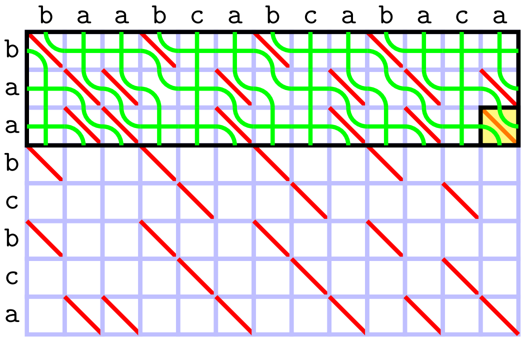
The seaweed algorithm

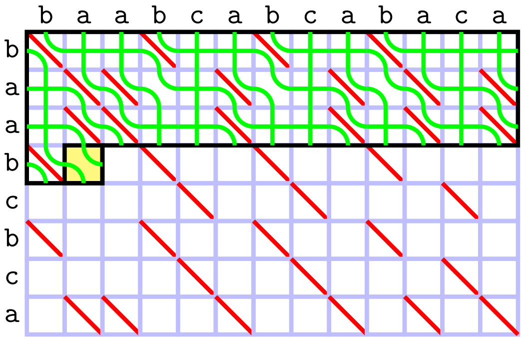
The seaweed algorithm



The seaweed algorithm

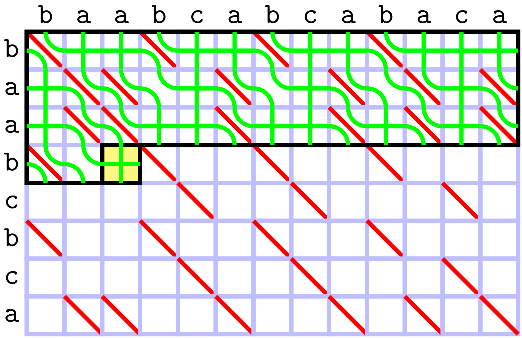
The seaweed algorithm




b a a b c a b c a b a c a
b
a
a
b
c
b
c
a

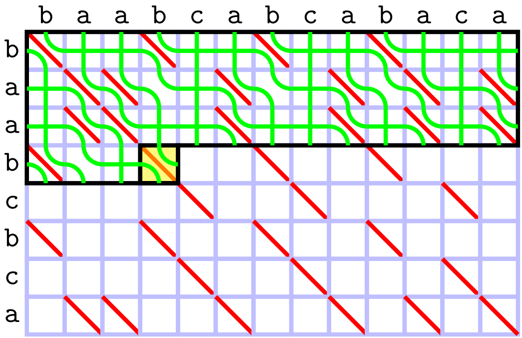
The seaweed algorithm

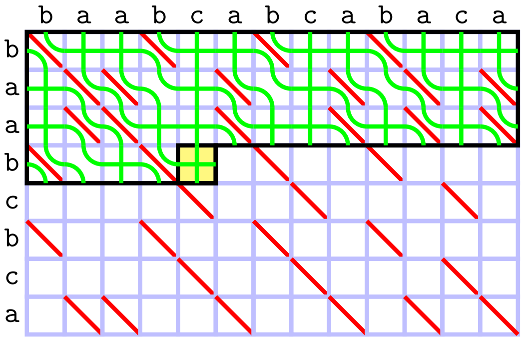
The seaweed algorithm



The seaweed algorithm

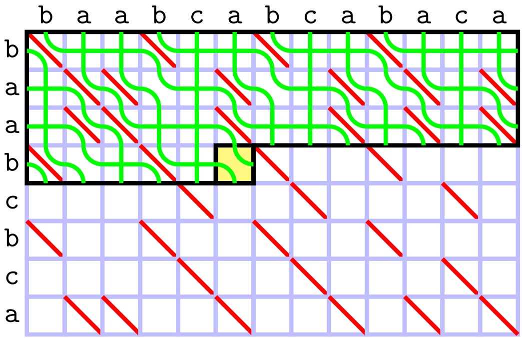
The seaweed algorithm




b a a b c a b c a b a c a
b a a b c b c a

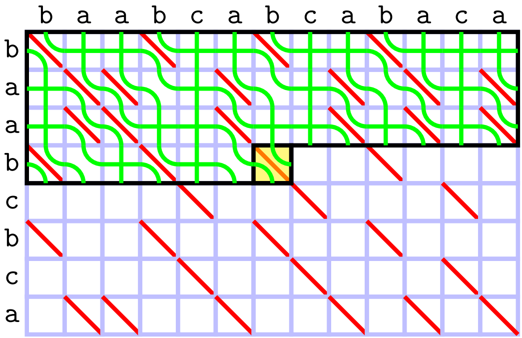
The seaweed algorithm

The seaweed algorithm



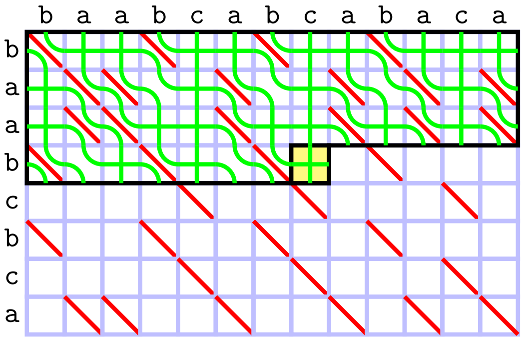
The seaweed algorithm

The seaweed algorithm



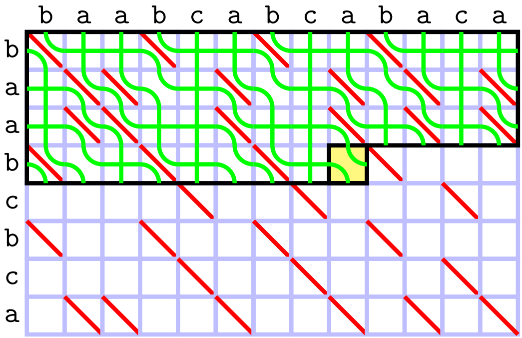
The seaweed algorithm

The seaweed algorithm



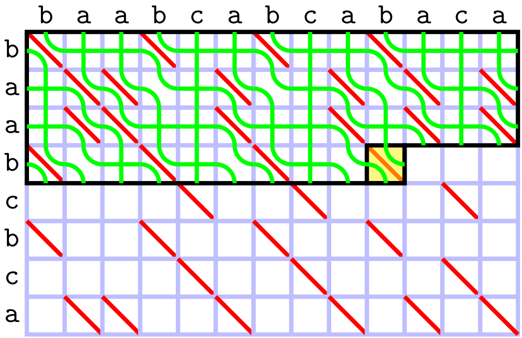
The seaweed algorithm

The seaweed algorithm



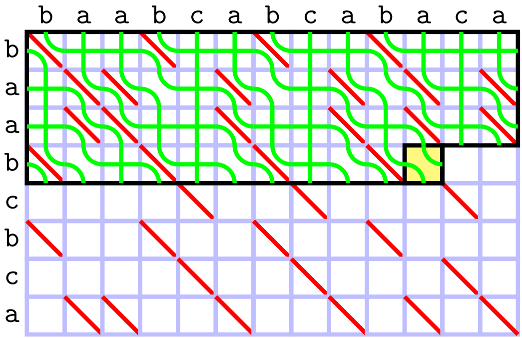
The seaweed algorithm

The seaweed algorithm



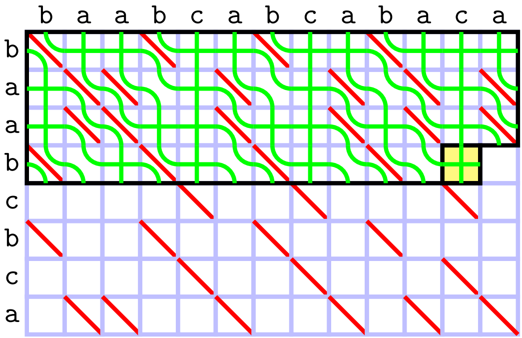
The seaweed algorithm

The seaweed algorithm



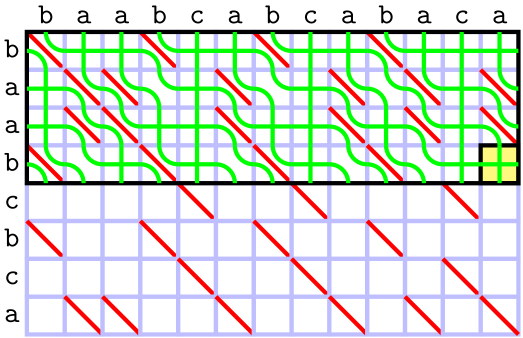
The seaweed algorithm

The seaweed algorithm



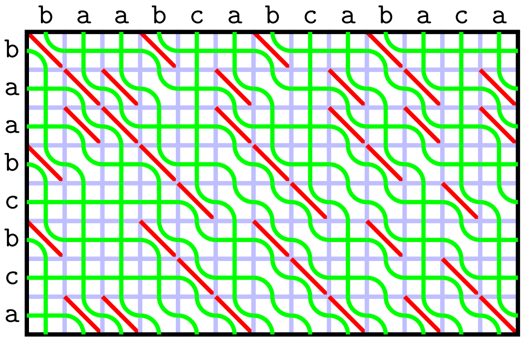
The seaweed algorithm

The seaweed algorithm



The seaweed algorithm

The seaweed algorithm



Semi-local LCS: the seaweed algorithm [T: 2006]

Iterate over alignment graph, tracing seaweeds

Pick cells in any order, respecting dependencies

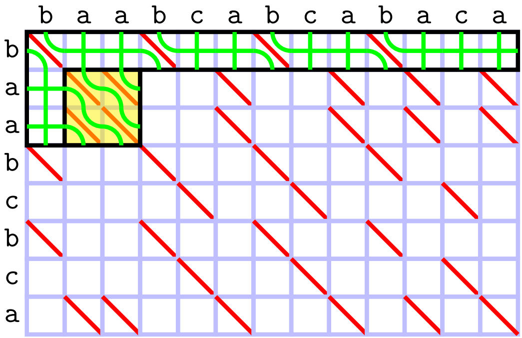
In every cell, the two entering seaweeds

- cross, if mismatch and they have not crossed before
- bend otherwise

Running time $O(mn)$

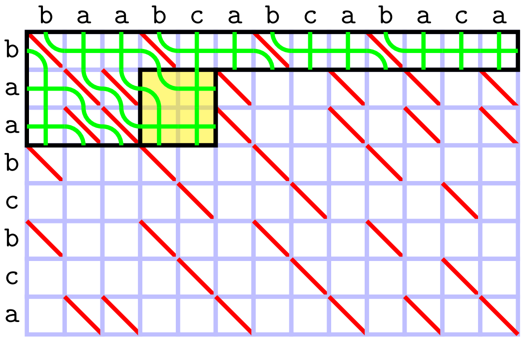
The seaweed algorithm

The micro-block seaweed algorithm



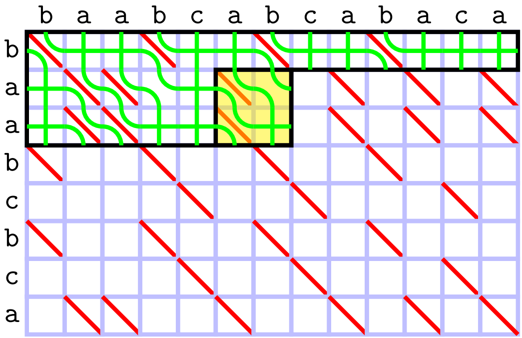
The seaweed algorithm

The micro-block seaweed algorithm



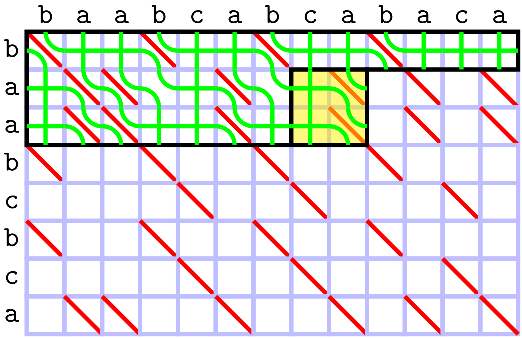
The seaweed algorithm

The micro-block seaweed algorithm



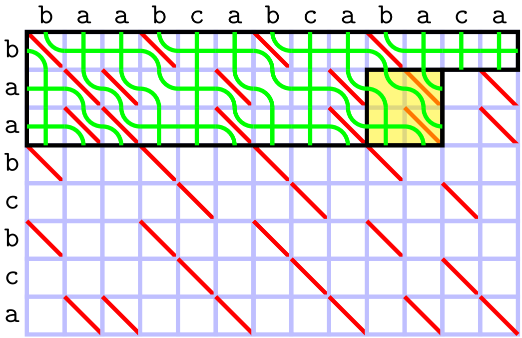
The seaweed algorithm

The micro-block seaweed algorithm



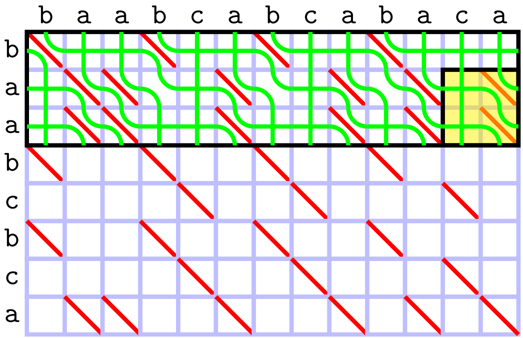
The seaweed algorithm

The micro-block seaweed algorithm



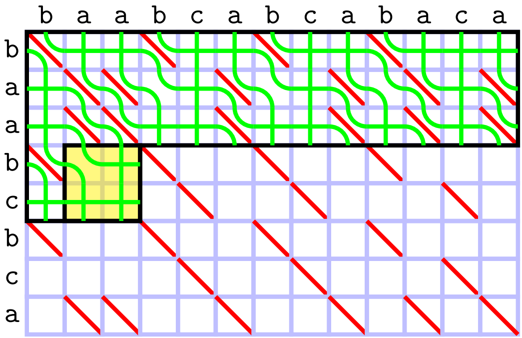
The seaweed algorithm

The micro-block seaweed algorithm



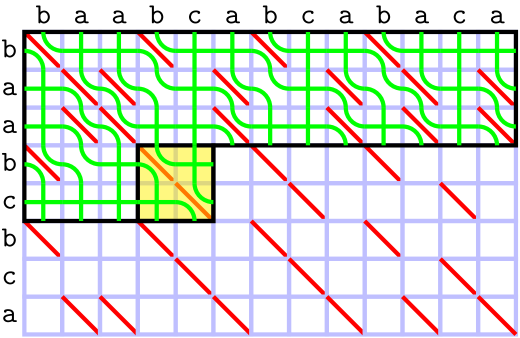
The seaweed algorithm

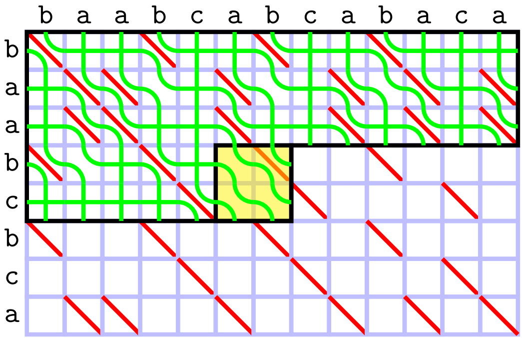
The micro-block seaweed algorithm



The seaweed algorithm

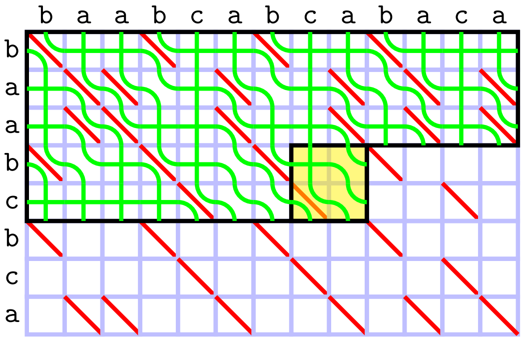
The micro-block seaweed algorithm

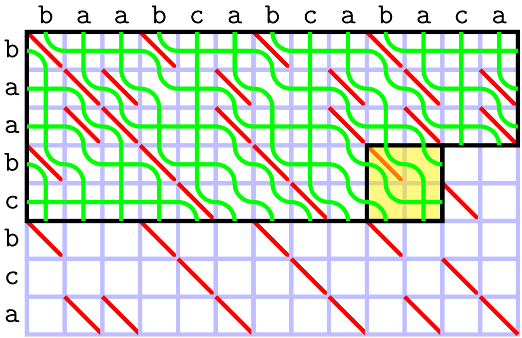



b a a b c a b c a b a c a
b
a
a
b
c
b
c
a

The seaweed algorithm

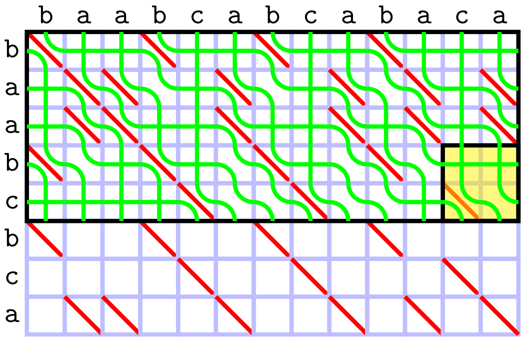
The micro-block seaweed algorithm




b a a b c a b c a b a c a
b
a
a
b
c
b
c
a

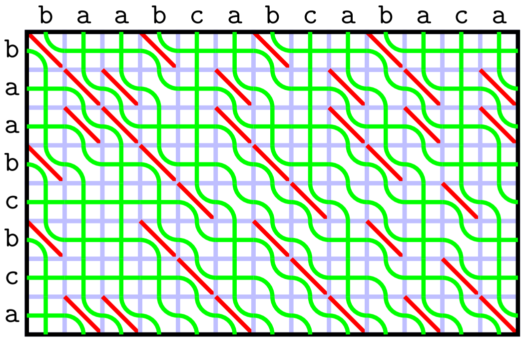
The seaweed algorithm

The micro-block seaweed algorithm



The seaweed algorithm

The micro-block seaweed algorithm



Semi-local LCS: the micro-block seaweed algorithm [T: 2007]

Iterate over alignment graph in blocks, tracing seaweeds

Use precomputed mapping of all possible block inputs to outputs

Block size: $t = O\left(\frac{\log n}{\log\log n}\right)$

Block interface: $O(t)$ values (input chars and seaweeds), each of size $O(\log n)$ but can be compressed to $O(\log\log n)$ by a recursive scheme

Running time $O\left(\frac{m}{t} \cdot \frac{n}{t} \cdot t \log\log n\right) = O\left(\frac{mn(\log\log n)^2}{\log n}\right)$, even on log-cost RAM

The cyclic LCS problem

Give the maximum LCS score for a vs all cyclic rotations of b

The cyclic LCS problem

Give the maximum LCS score for a vs all cyclic rotations of b

Cyclic LCS: running time

$O\left(\frac{mn^2}{\log n}\right)$	naive
$O(mn \log m)$	[Maes: 1990]
$O(mn)$	[Bunke, Bühler: 1993; Landau+: 1998; Schmidt: 1998]
$O\left(\frac{mn(\log\log n)^2}{\log n}\right)$	[T: 2007]

Cyclic LCS: the algorithm

Run the micro-block seaweed algorithm on a vs bb, time $O\left(\frac{mn(\log\log n)^2}{\log n}\right)$

Make n string-substring LCS queries, time negligible

The longest repeating subsequence problem

Find the longest subsequence of a that is a square (a repetition of two identical strings)

Motivated by "tandem repeats" in genome

The longest repeating subsequence problem

Find the longest subsequence of a that is a square (a repetition of two identical strings)

Motivated by "tandem repeats" in genome

Longest repeating subsequence: running time

$O(n^3)$	naive
$O(n^2)$	[Kosowski: 2004]
$O\left(\frac{n^2(\log\log n)^2}{\log n}\right)$	[T: 2007]

Longest repeating subsequence: the algorithm

Run the micro-block seaweed algorithm on a vs bb, time $O\left(\frac{mn(\log\log n)^2}{\log n}\right)$

Make $n-1$ suffix-prefix LCS queries, time negligible

The approximate pattern matching problem

Give the substring closest to a by alignment score, starting at each position in b

Assume rational alignment score

Approximate pattern matching: running time

$O(mn)$		[Sellers: 1980]
$O\left(\frac{mn}{\log n}\right)$	$\sigma = O(1)$	via [Masek, Paterson: 1980]
$O\left(\frac{mn(\log\log n)^2}{\log n}\right)$		via [Paterson, Dancik: 1994]

Approximate pattern matching: the algorithm

Run the micro-block seaweed algorithm on a vs b under given alignment score in time $O\left(\frac{mn(\log\log n)^2}{\log n}\right)$

The implicit semi-local edit score matrix:

- an anti-Monge matrix
- approximate pattern matching $\sim$ row minima

Row minima in $O(n)$ element queries [Aggarwal+: 1987]

Each query in time $O(\log^2 n)$ using the range tree representation, combined query time negligible

Overall running time dominated by block seaweed algorithm, same as [Paterson, Dancik: 1994]

The periodic string-substring LCS problem

Give (implicit) LCS scores for a vs each substring of $b = \ldots uuu \ldots = u^{\pm\infty}$

Let u be of length p; may assume that every character of a occurs in u

The tandem LCS problem

Give LCS score for a vs $b = u^k$

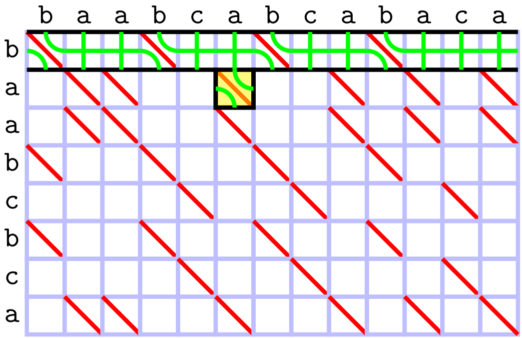
We have $n = kp$; may assume $k \leq m$ (otherwise LCS score is m)

Tandem LCS: running time

$O(mkp)$	naive
$O(m(k+p))$	[Landau, Ziv-Ukelson: 2001]
$O(mp)$	[NEW]

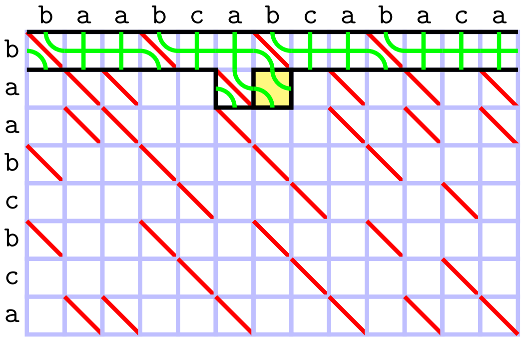
The seaweed algorithm

The periodic seaweed algorithm



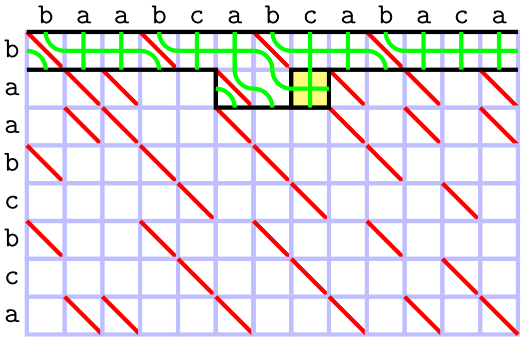
The seaweed algorithm

The periodic seaweed algorithm



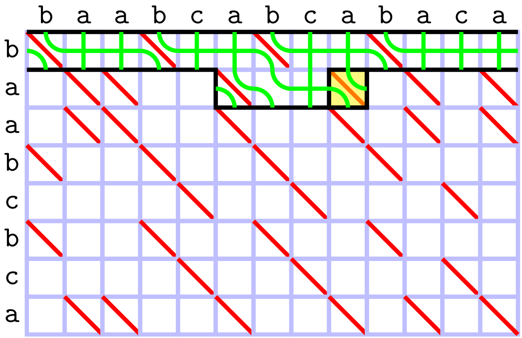
The seaweed algorithm

The periodic seaweed algorithm



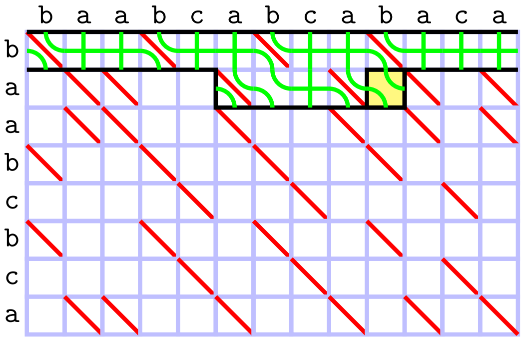
The seaweed algorithm

The periodic seaweed algorithm



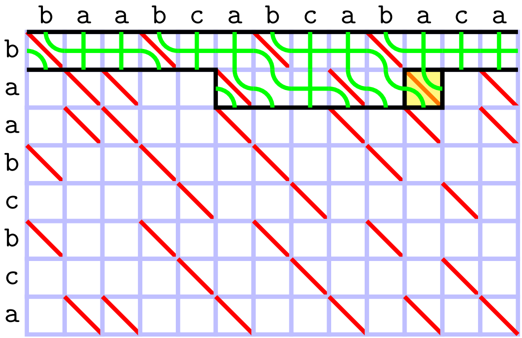
The seaweed algorithm

The periodic seaweed algorithm



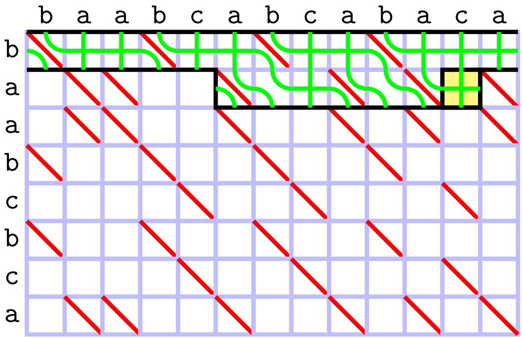
The seaweed algorithm

The periodic seaweed algorithm



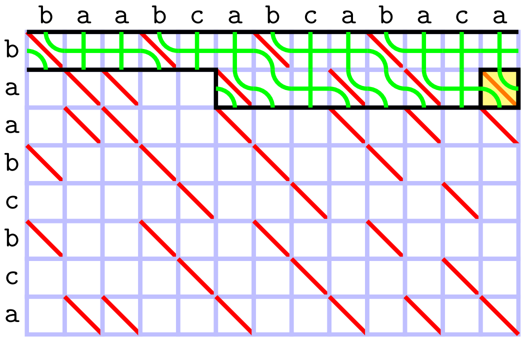
The seaweed algorithm

The periodic seaweed algorithm



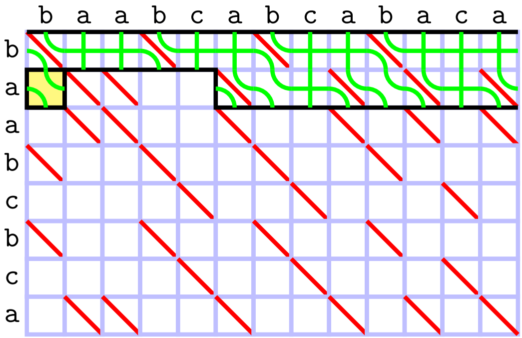
The seaweed algorithm

The periodic seaweed algorithm



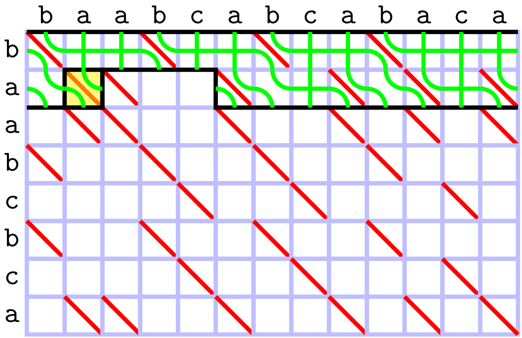
The seaweed algorithm

The periodic seaweed algorithm



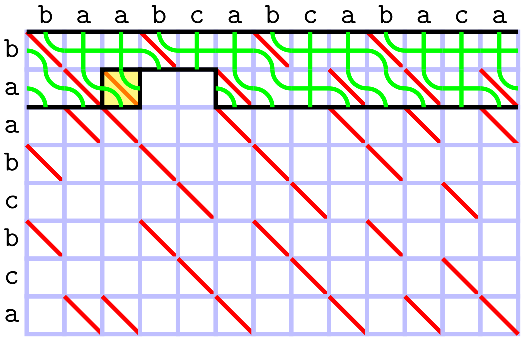
The seaweed algorithm

The periodic seaweed algorithm



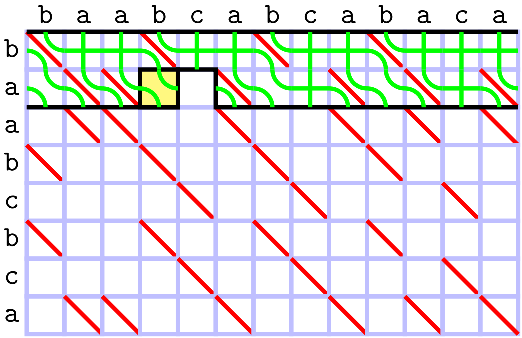
The seaweed algorithm

The periodic seaweed algorithm



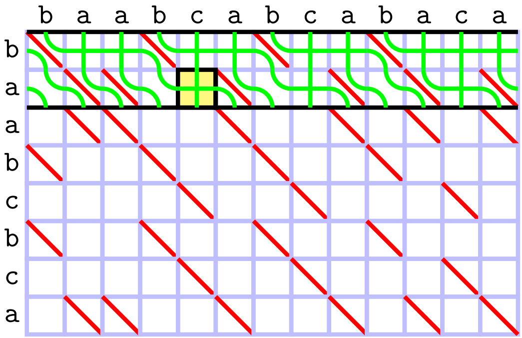
The seaweed algorithm

The periodic seaweed algorithm



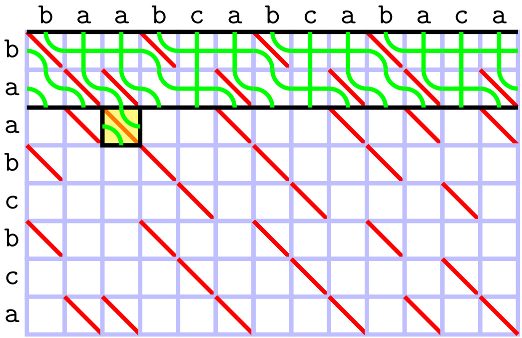
The seaweed algorithm

The periodic seaweed algorithm



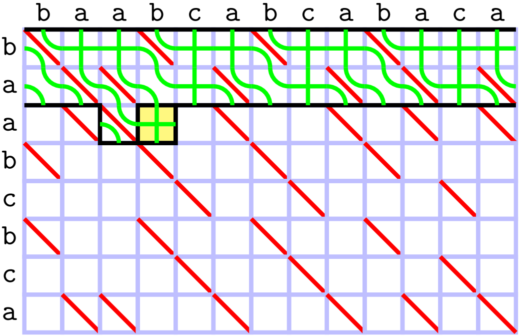
The seaweed algorithm

The periodic seaweed algorithm



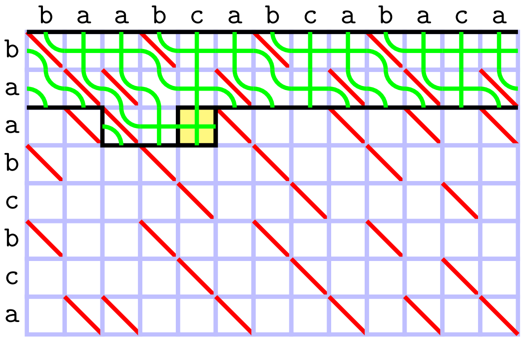
The seaweed algorithm

The periodic seaweed algorithm



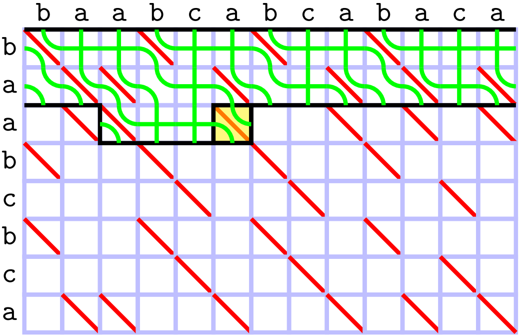
The seaweed algorithm

The periodic seaweed algorithm



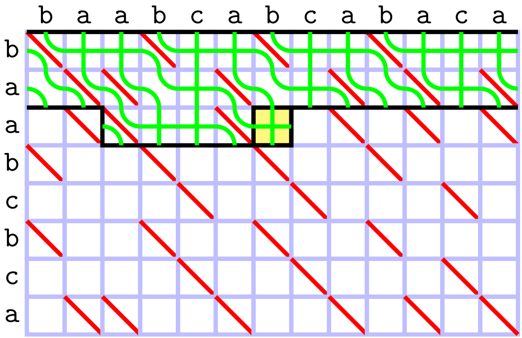
The seaweed algorithm

The periodic seaweed algorithm



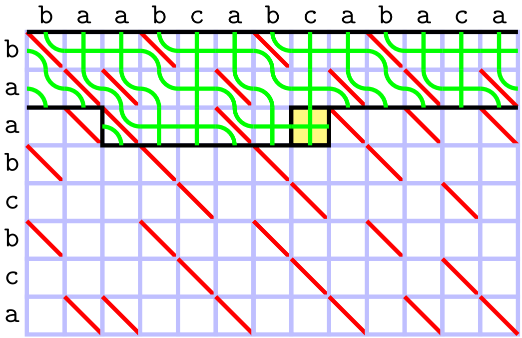
The seaweed algorithm

The periodic seaweed algorithm



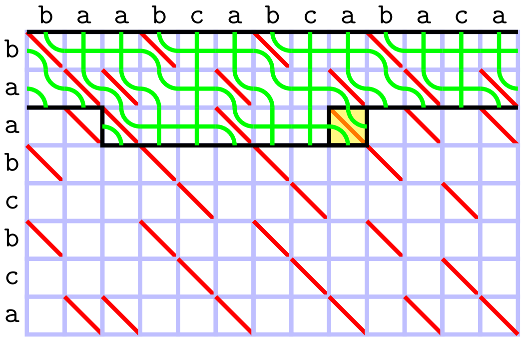
The seaweed algorithm

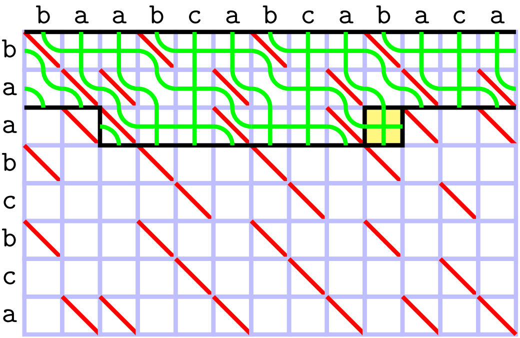
The periodic seaweed algorithm



The seaweed algorithm

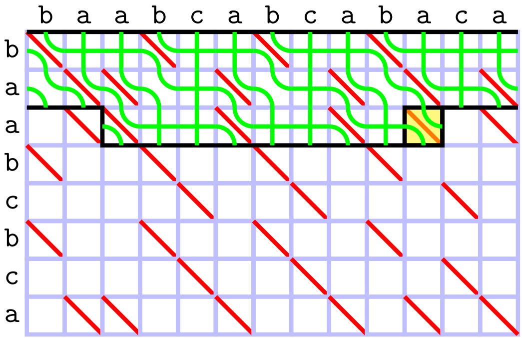
The periodic seaweed algorithm




b a a b c a b c a b a c a
b
a
a
b
c
b
c
a

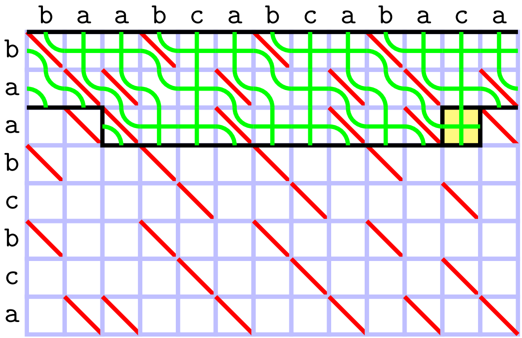
The seaweed algorithm

The periodic seaweed algorithm



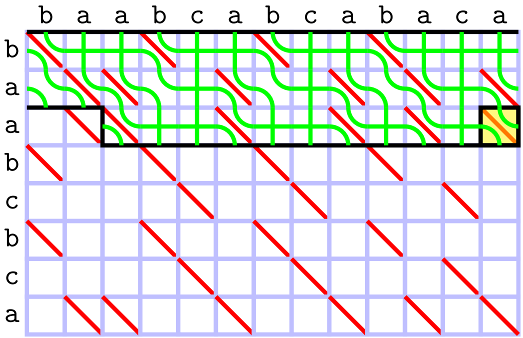
The seaweed algorithm

The periodic seaweed algorithm



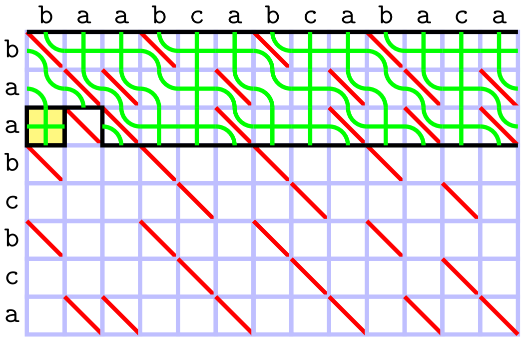
The seaweed algorithm

The periodic seaweed algorithm



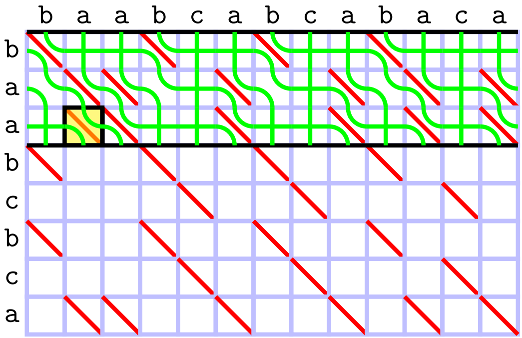
The seaweed algorithm

The periodic seaweed algorithm



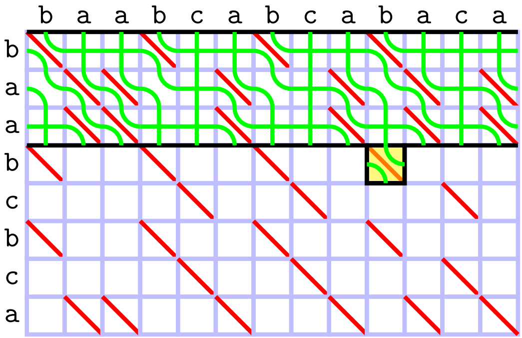
The seaweed algorithm

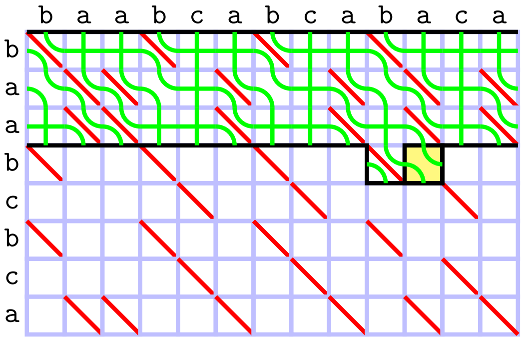
The periodic seaweed algorithm



The seaweed algorithm

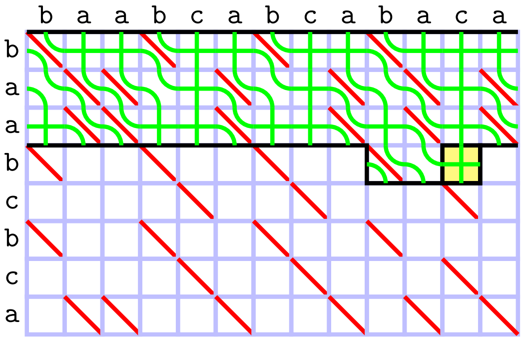
The periodic seaweed algorithm




b a a b c a b c a b a c a
b
a
a
b
c
b
c
a

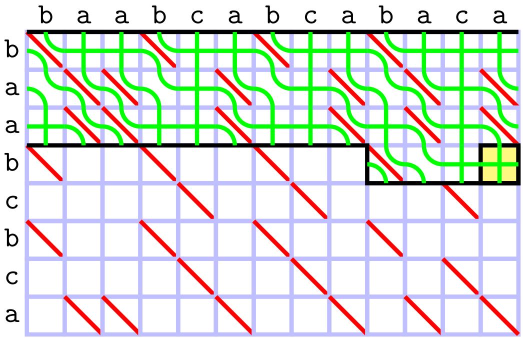
The seaweed algorithm

The periodic seaweed algorithm



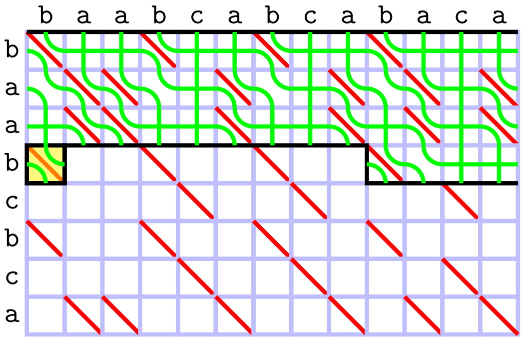
The seaweed algorithm

The periodic seaweed algorithm



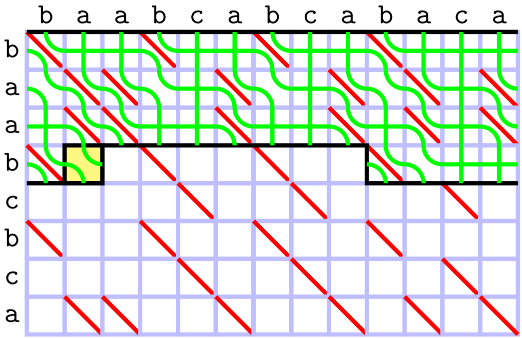
The seaweed algorithm

The periodic seaweed algorithm



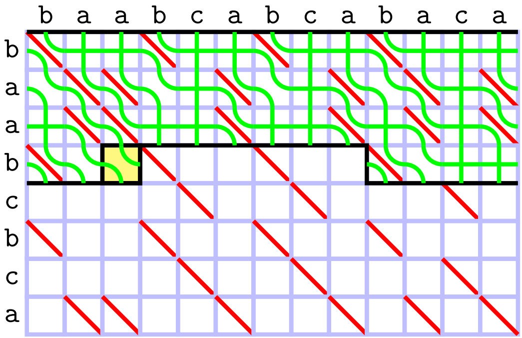
The seaweed algorithm

The periodic seaweed algorithm



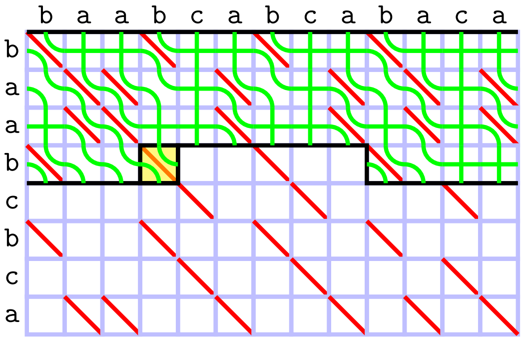
The seaweed algorithm

The periodic seaweed algorithm



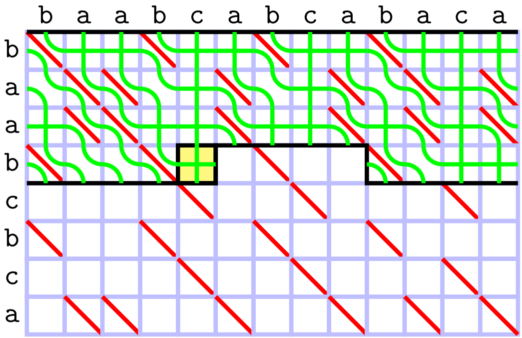
The seaweed algorithm

The periodic seaweed algorithm



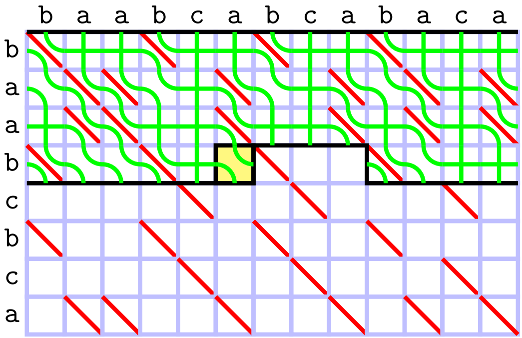
The seaweed algorithm

The periodic seaweed algorithm



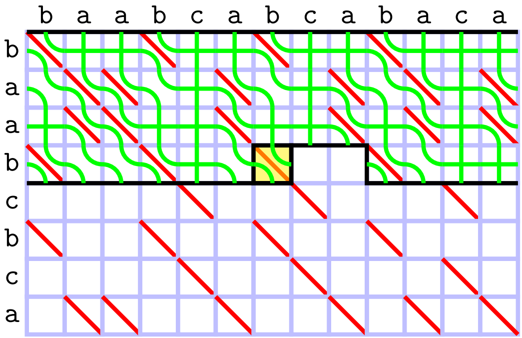
The seaweed algorithm

The periodic seaweed algorithm



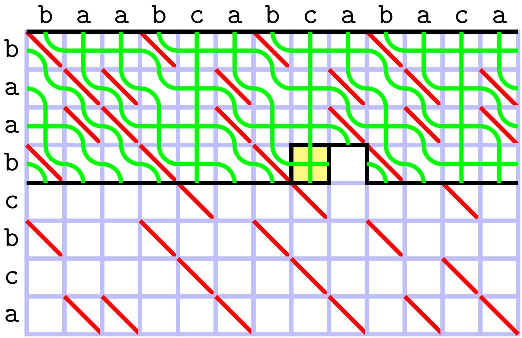
The seaweed algorithm

The periodic seaweed algorithm



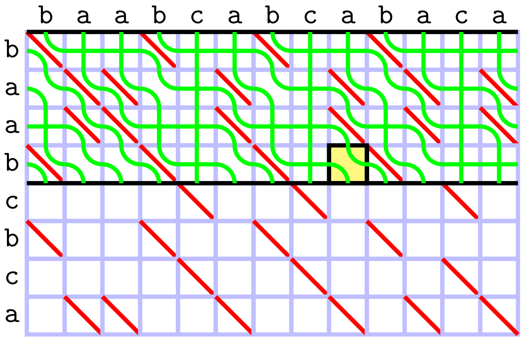
The seaweed algorithm

The periodic seaweed algorithm



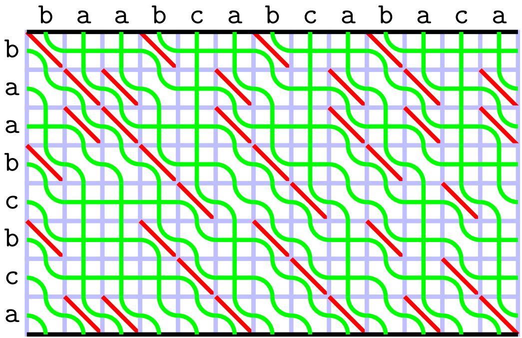
The seaweed algorithm

The periodic seaweed algorithm



The seaweed algorithm

The periodic seaweed algorithm



Periodic string-substring LCS: The periodic seaweed algorithm

Iterate over alignment graph, tracing seaweeds row-by-row

In every row, start from a match cell and move rightwards, wrapping around at the graph edge

In every cell, the two entering seaweeds

- cross, if mismatch and they have not crossed before
- bend otherwise
- at right edge of the graph, wrap around back to left edge

Running time $O(mn)$

Querying a string-substring LCS score (including tandem LCS): count each nonzero dominated by query point with appropriate multiplicity, either directly or via a range tree

The tandem alignment problem

Give the substring closest to a by alignment score among certain substrings of $b = u^{\pm\infty}$:

- global: substrings of the form b^k across all k
- cyclic: substrings of length kp across all k
- local: substrings of any length

Tandem alignment: running time

$O(m^2 p)$	all	naive
$O(mp)$	global	[Myers, Miller: 1989]
$O(mp \log p)$	cyclic	[Benson: 2005]
$O(mp)$	cyclic	[NEW]
$O(mp)$	local	[Myers, Miller: 1989]

Cyclic tandem alignment: the algorithm

Run periodic seaweed algorithm (under given alignment score), time $O(np)$

For each $k \in [1 : m]$:

- solve tandem LCS (under given alignment score) for a against b^k
- obtain p successive string-substring alignment scores by inremental score updating, each in time $O(1)$

Running time $O(mp)$

Conclusions and future work

Semi-local LCS problem:

- representation by implicit unit-Monge matrices
- generalisation to rational alignment scores
- open: real alignment scores?

The seaweed and micro-block seaweed algorithms:

- a simple algorithm for semi-local LCS
- semi-local LCS in time $o(mn)$ via micro-blocks
- improvements on related problems

The periodic seaweed algorithm:

- a straightforward extension of the seaweed algorithm
- periodic semi-local LCS in time $O(mp)$
- natural applications
- open: $o(mp)$ via micro-blocks?