Approximate matching in grammar-compressed strings

Alexander Tiskin

Department of Computer Science, University of Warwick
http://go.warwick.ac.uk/alextiskin

Introduction

String matching: finding an exact pattern in a string

String comparison: finding similar patterns in two strings

Introduction

String matching: finding an exact pattern in a string

String comparison: finding similar patterns in two strings

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

Closely related to approximate string matching: whole pattern string vs substrings in text string

Introduction

Overview

Standard approach to string comparison: dynamic programming

Our approach: the algebra of unit-Monge matrices, a.k.a. seaweed braids

Can be used either for dynamic programming, or for divide-and-conquer

Divide-and-conquer is better for:

- comparing strings dynamically (truncation, concatenation)
- comparing compressed srtrings (e.g. LZ-compression)
- comparing strings in parallel

The "conquer" step is non-trivial — but not so much that it couldn't have been discovered much earlier!

The "squared paper" notation:

Integers $\{\ldots -2, -1, 0, 1, 2, \ldots\}$ $\quad x^- = x - \frac{1}{2}$ $\quad x^+ = x + \frac{1}{2}$

Half-integers $\{\ldots -\frac{3}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \ldots\} = \{\ldots (-2)^+, (-1)^+, 0^+, 1^+, 2^+\}$

Planar dominance:

- $(i,j) \ll (i',j')$ iff $i < i'$ and $j < j'$ (the "above-left" relation)
- $(i,j) \gg (i',j')$ iff $i > i'$ and $j < j'$ (the "below-left" relation)

where "above/below" follow matrix convention (not the Cartesian one!)

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

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Given matrix D, its distribution matrix is made up of $\gtrless$-dominance sums: $D^{\Sigma}(i,j) = \sum_{\hat{\imath}>i,\hat{\jmath}<j} D(i,j)$

Given matrix D, its distribution matrix is made up of $\gtrless$-dominance sums:
$D^{\Sigma}(i,j) = \sum_{\hat{\imath}>i,\hat{\jmath}<j} D(i,j)$

Given matrix E, its density matrix is made up of quadrangle differences:
$E^{\square}(\hat{\imath},\hat{\jmath}) = E(\hat{\imath}^{-},\hat{\jmath}^{+}) - E(\hat{\imath}^{-},\hat{\jmath}^{-}) - E(\hat{\imath}^{+},\hat{\jmath}^{+}) + E(\hat{\imath}^{+},\hat{\jmath}^{-})$

where D^{Σ}, E over integers; D, $E^{\square}$ over half-integers

Given matrix D, its distribution matrix is made up of $\gtrless$-dominance sums:
$D^{\Sigma}(i,j) = \sum_{\hat{\imath}>i,\hat{\jmath}<j} D(i,j)$

Given matrix E, its density matrix is made up of quadrangle differences:
$E^{\square}(\hat{\imath},\hat{\jmath}) = E(\hat{\imath}^{-},\hat{\jmath}^{+}) - E(\hat{\imath}^{-},\hat{\jmath}^{-}) - E(\hat{\imath}^{+},\hat{\jmath}^{+}) + E(\hat{\imath}^{+},\hat{\jmath}^{-})$

where D^{Σ}, E over integers; D, $E^{\square}$ over half-integers

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{\Sigma} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}^{\square} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Given matrix D, its distribution matrix is made up of $\gtrdot$-dominance sums:
$D^{\Sigma}(i,j) = \sum_{\hat{\imath}>i,\hat{\jmath}<j} D(i,j)$

Given matrix E, its density matrix is made up of quadrangle differences:
$E^{\square}(\hat{\imath},\hat{\jmath}) = E(\hat{\imath}^{-},\hat{\jmath}^{+}) - E(\hat{\imath}^{-},\hat{\jmath}^{-}) - E(\hat{\imath}^{+},\hat{\jmath}^{+}) + E(\hat{\imath}^{+},\hat{\jmath}^{-})$

where D^{Σ}, E over integers; D, $E^{\square}$ over half-integers

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{\Sigma} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}^{\square} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

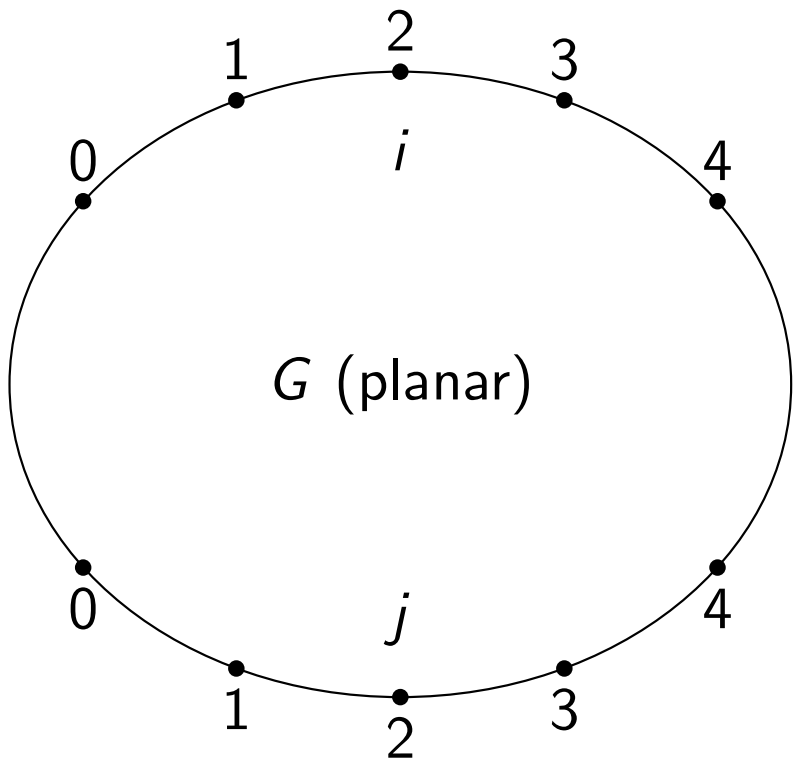
$(D^{\Sigma})^{\square} = D$ for all D

Matrix E is simple, if $(E^{\square})^{\Sigma} = E$; equivalently, if it has all zeros in the left column and bottom row

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

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

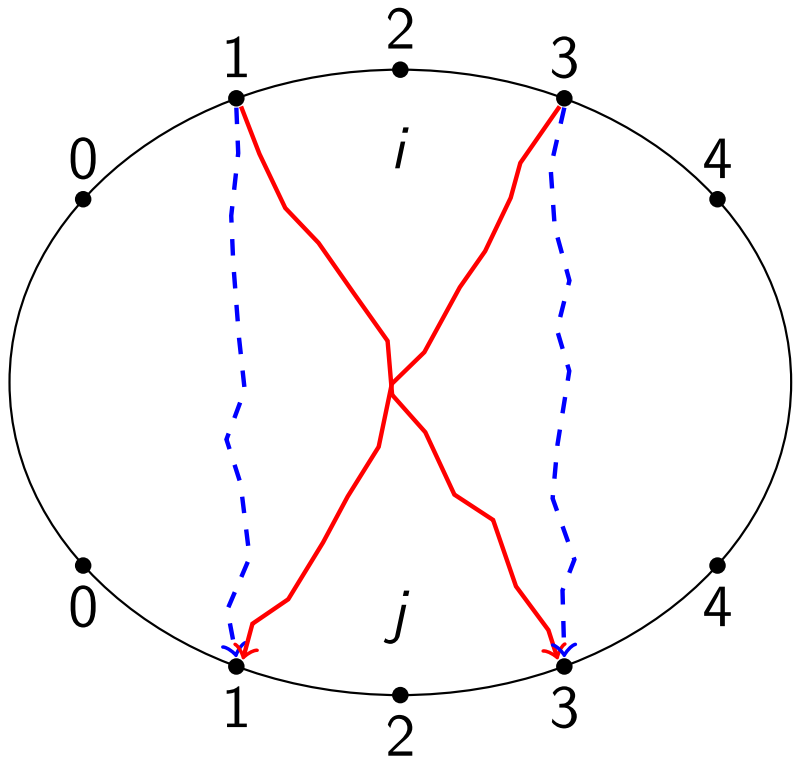
G. Monge (1746–1818)



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

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

G. Monge (1746–1818)



$$blue + blue \leq red + red$$

Introduction

Terminology and notation

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

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

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

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

Simple unit-Monge matrix: P^{Σ}, where P is a permutation matrix

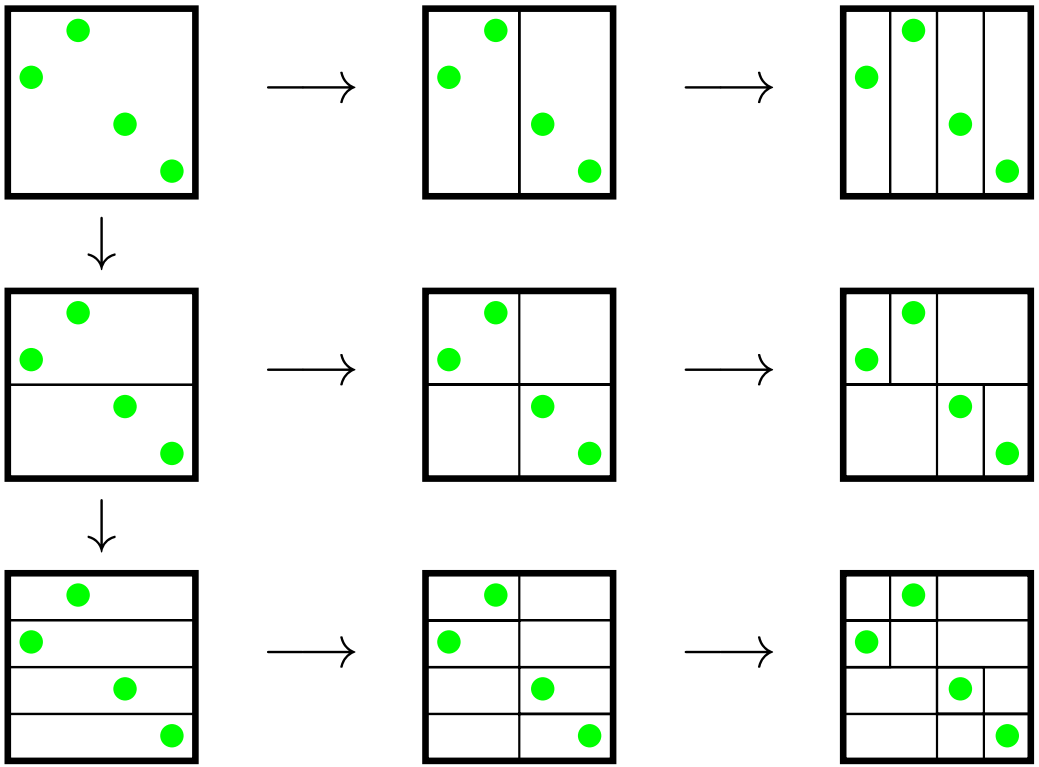
P^{Σ} sometimes called the rank function of P

Seaweed matrix: P used as an implicit representation of P^{Σ}

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{\Sigma} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

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

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



Efficient P^{Σ} queries: (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

A P^{Σ} query is equivalent to $\gtrless$-dominance counting: how many nonzeros are $\gtrless$-dominated by query point?

Answer: sum up nonzero counts in $\leq \log^2 n$ disjoint canonical ranges

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

Efficient P^{Σ} queries: (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

A P^{Σ} query is equivalent to $\gtrless$-dominance counting: how many nonzeros are $\gtrless$-dominated by query point?

Answer: sum up nonzero counts in $\leq \log^2 n$ 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

Total size $O(n)$, query time $O\left(\frac{\log n}{\log \log n}\right)$ [JáJá+: 2004]
[Chan, Pătraşcu: 2010]

Matrix distance multiplication

Seaweed braids

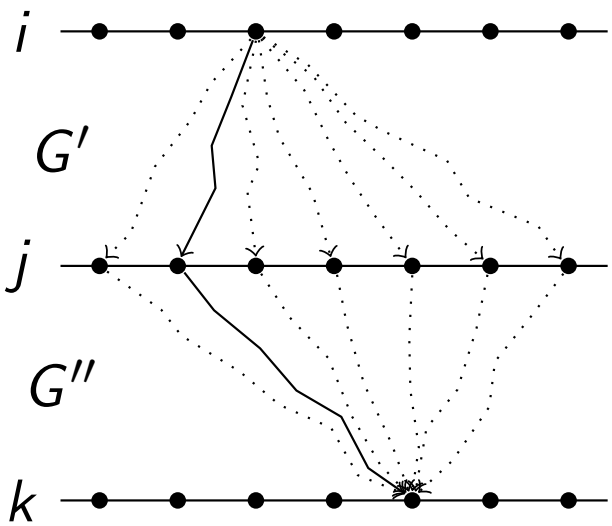
Distance algebra (a.k.a $(\min, +)$ or tropical algebra):

- addition $\oplus$ given by min, multiplication $\odot$ given by $+$

Matrix $\odot$-multiplication

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

Intuition: gluing distances in a union of graphs



Matrix distance multiplication

Seaweed braids

Matrix classes closed under $\odot$-multiplication (for given n):

- general (integer, real) matrices $\sim$ general weighted graphs
- Monge matrices $\sim$ planar weighted graphs
- simple unit-Monge matrices $\sim$ grid-like graphs

Implicit $\odot$-multiplication: define $P_A \boxdot P_B = P_C$ as $P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$

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

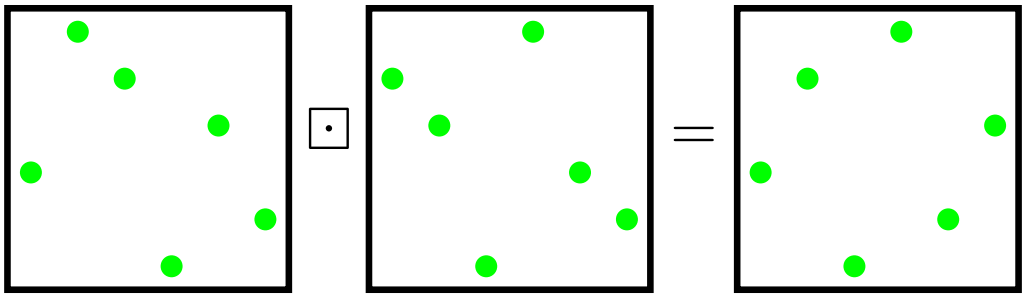
- simple unit-Monge matrices under $\odot$
- equivalently, permutation (seaweed) matrices under $\boxdot$

Also known as the 0-Hecke monoid of the symmetric group $H_0(\mathcal{S}_n)$

Matrix distance multiplication

Seaweed braids

$P_A \boxdot P_B = P_C$ can be seen as combing of seaweed braids

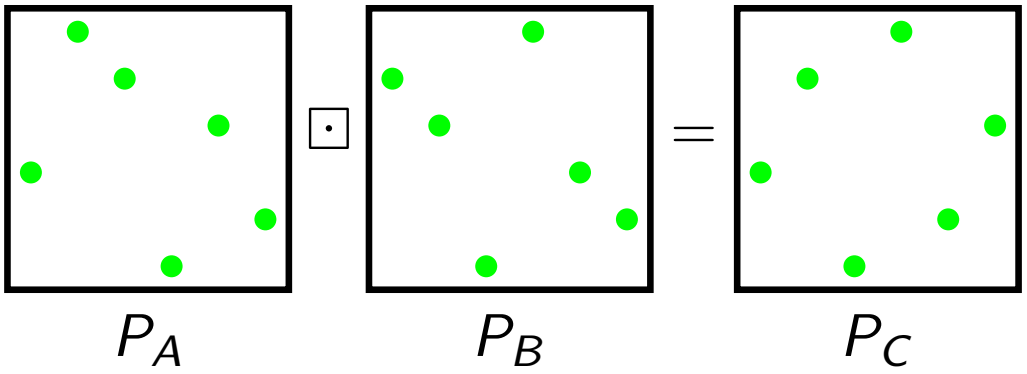


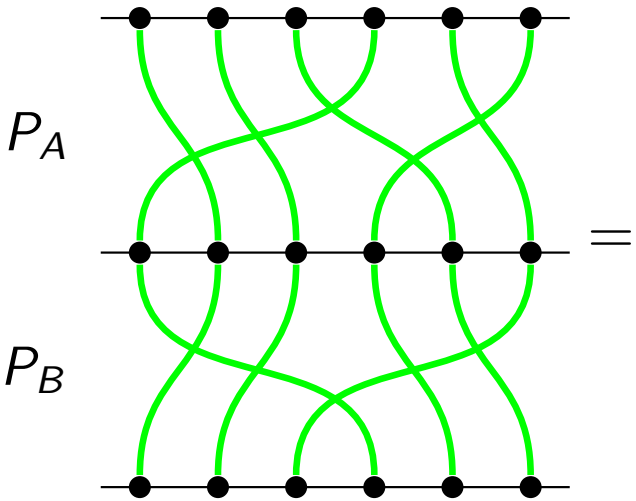
P_A P_B P_C

Matrix distance multiplication

Seaweed braids

$P_A \boxdot P_B = P_C$ can be seen as combing of seaweed braids

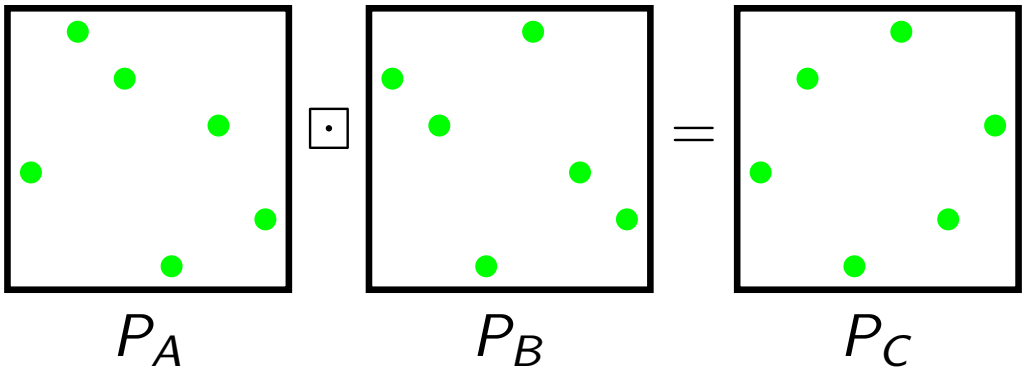


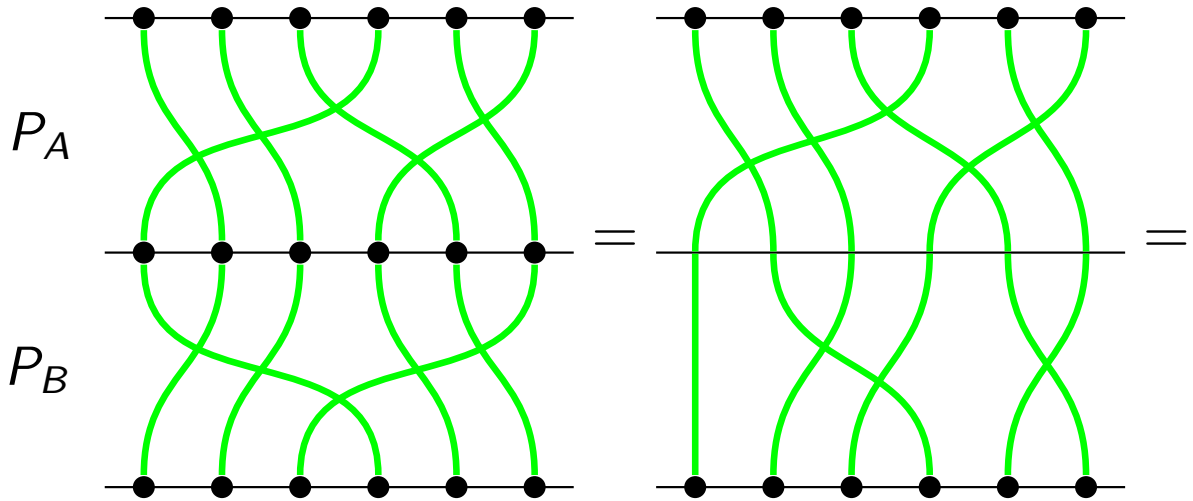


Matrix distance multiplication

Seaweed braids

$P_A \boxdot P_B = P_C$ can be seen as combing of seaweed braids

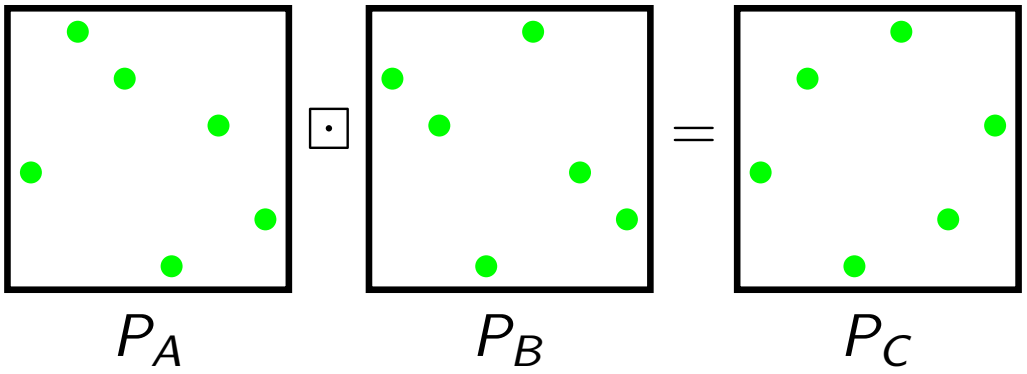


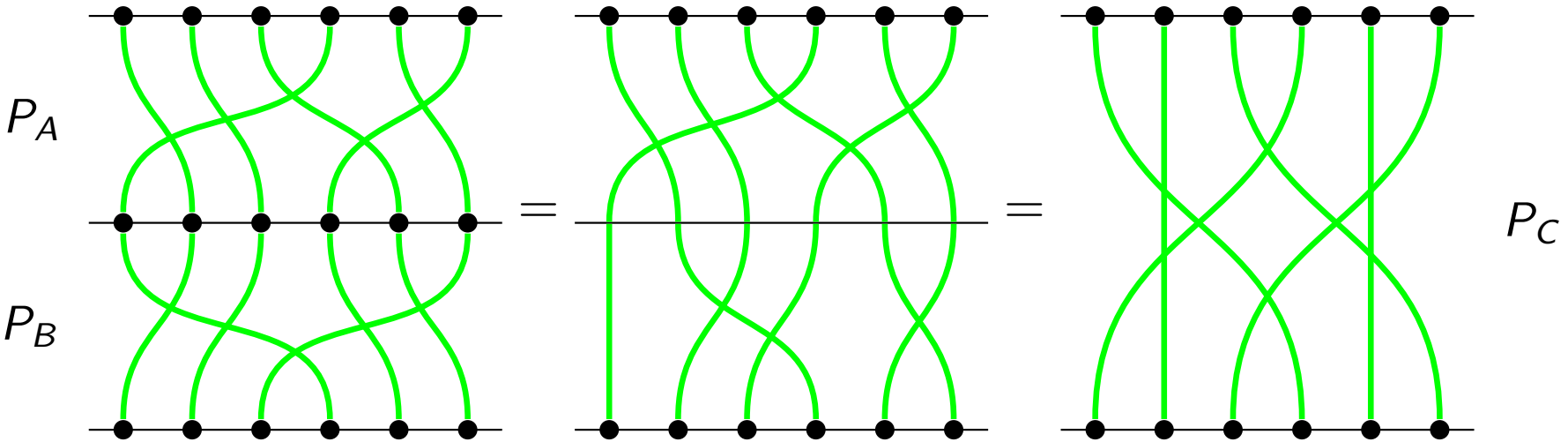


Matrix distance multiplication

Seaweed braids

$P_A \boxdot P_B = P_C$ can be seen as combing of seaweed braids





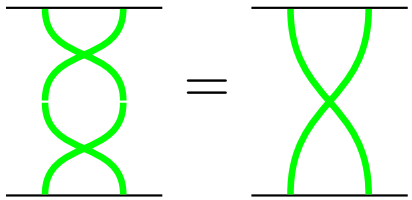
Matrix distance multiplication

Seaweed braids

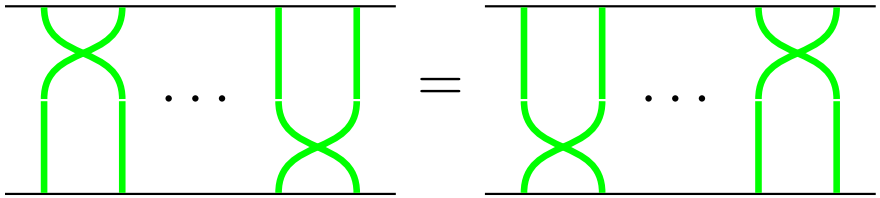
The seaweed monoid $\mathcal{T}_n$:

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

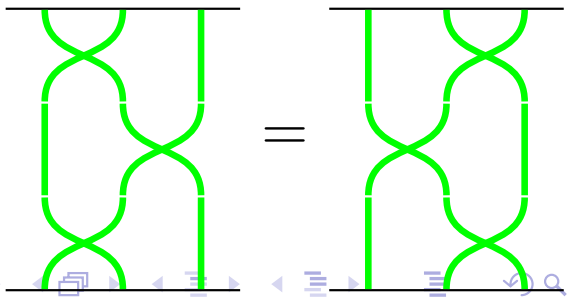
Idempotence:
$g_i^2 = g_i \quad$ for all i



Far commutativity:
$g_i g_j = g_j g_i \quad j - i > 1$



Braid relations:
$g_i g_j g_i = g_j g_i g_j \quad j - i = 1$

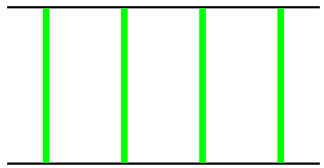


Matrix distance multiplication

Seaweed braids

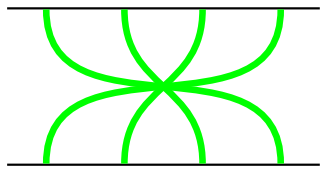
Identity: $1 \boxdot x = x$

$$1 = \begin{bmatrix} \bullet & \cdot & \cdot & \cdot \\ \cdot & \bullet & \cdot & \cdot \\ \cdot & \cdot & \bullet & \cdot \\ \cdot & \cdot & \cdot & \bullet \end{bmatrix} =$$



Zero: $0 \boxdot x = 0$

$$0 = \begin{bmatrix} \cdot & \cdot & \cdot & \bullet \\ \cdot & \cdot & \bullet & \cdot \\ \cdot & \bullet & \cdot & \cdot \\ \bullet & \cdot & \cdot & \cdot \end{bmatrix} =$$



Matrix distance multiplication

Seaweed braids

Related structures:

- positive braids: far comm; braid relations
- braids: $g_i g_i^{-1} = 1$; far comm; braid relations
- Coxeter's presentation of $\mathcal{S}_n$: $g_i^2 = 1$; far comm; braid relations
- locally free idempotent monoid: idem; far comm [Vershik+: 2000]

Generalisations:

- general 0-Hecke monoids [Fomin, Greene: 1998; Buch+: 2008]
- Coxeter monoids [Tsaranov: 1990; Richardson, Springer: 1990]
- $\mathcal{J}$-trivial monoids [Denton+: 2011]

Matrix distance multiplication

Seaweed braids

Computation in the seaweed monoid: a confluent rewriting system can be obtained by software (SEMIGROUPE, GAP)

Matrix distance multiplication

Seaweed braids

Computation in the seaweed monoid: a confluent rewriting system can be obtained by software (SEMIGROUPE, GAP)

$\mathcal{T}_3$: 1, $a = g_1$, $b = g_2$; ab, ba; $aba = 0$

$aa \to a$ $\quad$ $bb \to b$ $\quad$ $bab \to 0$ $\quad$ $aba \to 0$

Matrix distance multiplication

Seaweed braids

Computation in the seaweed monoid: a confluent rewriting system can be obtained by software (SEMIGROUPE, GAP)

$\mathcal{T}_3$: 1, $a = g_1$, $b = g_2$; ab, ba; $aba = 0$

$aa \to a$ $\quad$ $bb \to b$ $\quad$ $bab \to 0$ $\quad$ $aba \to 0$

$\mathcal{T}_4$: 1, $a = g_1$, $b = g_2$, $c = g_3$; ab, ac, ba, bc, cb, aba, abc, acb, bac, bcb, cba, $abac$, $abcb$, $acba$, $bacb$, $bcba$, $abacb$, $abcba$, $bacba$; $abacba = 0$

$aa \to a$ $\quad$ $ca \to ac$ $\quad$ $bab \to aba$ $\quad$ $cbac \to bcba$

$bb \to b$ $\quad$ $cc \to c$ $\quad$ $cbc \to bcb$ $\quad$ $abacba \to 0$

Matrix distance multiplication

Seaweed braids

Computation in the seaweed monoid: a confluent rewriting system can be obtained by software (SEMIGROUPE, GAP)

$\mathcal{T}_3$: 1, $a = g_1$, $b = g_2$; ab, ba; $aba = 0$

$aa \to a$ $\quad$ $bb \to b$ $\quad$ $bab \to 0$ $\quad$ $aba \to 0$

$\mathcal{T}_4$: 1, $a = g_1$, $b = g_2$, $c = g_3$; ab, ac, ba, bc, cb, aba, abc, acb, bac, bcb, cba, $abac$, $abcb$, $acba$, $bacb$, $bcba$, $abacb$, $abcba$, $bacba$; $abacba = 0$

$aa \to a$ $\quad$ $ca \to ac$ $\quad$ $bab \to aba$ $\quad$ $cbac \to bcba$

$bb \to b$ $\quad$ $cc \to c$ $\quad$ $cbc \to bcb$ $\quad$ $abacba \to 0$

Easy to use, but not an efficient algorithm

The implicit unit-Monge matrix $\odot$-multiplication problem

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

Matrix distance multiplication

Seaweed matrix multiplication

The implicit unit-Monge matrix $\odot$-multiplication problem

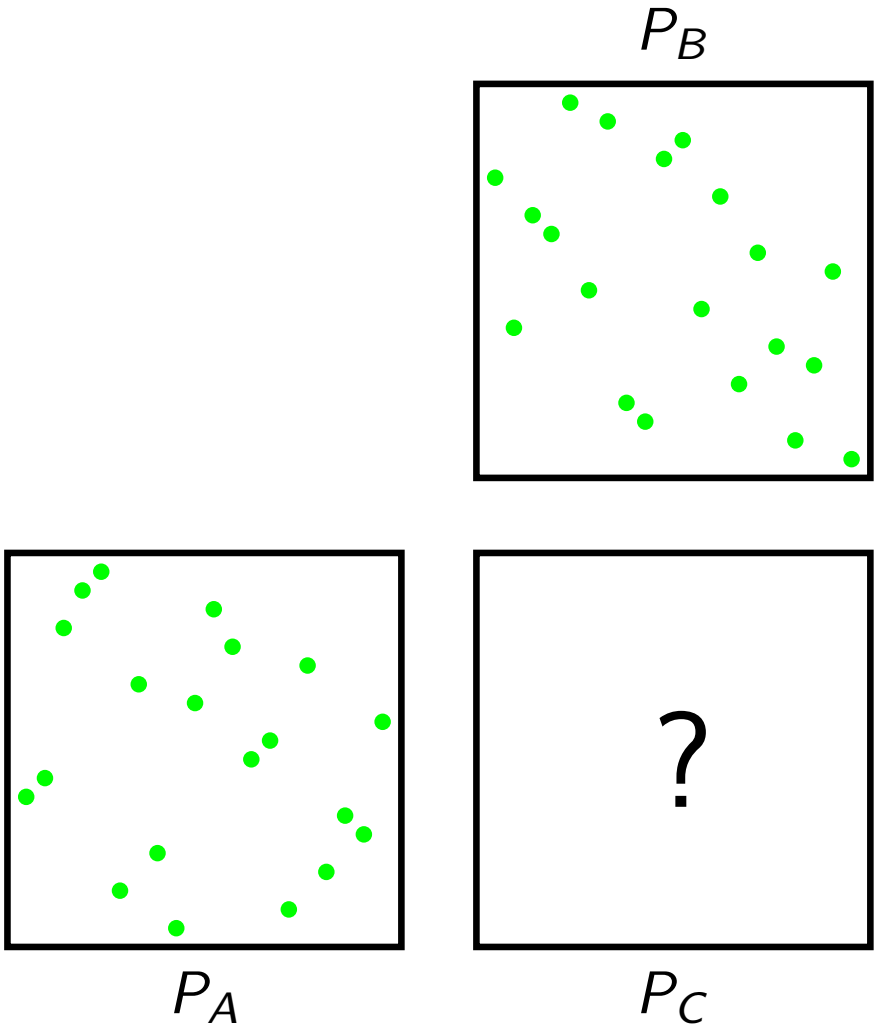
Given permutation matrices P_A, P_B, compute P_C, such that $P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$ (equivalently, $P_A \boxdot P_B = P_C$)

Matrix $\odot$-multiplication: running time

type	time	
general	$O(n^3)$	standard
	$O\left(\frac{n^3(\log\log n)^3}{\log^2 n}\right)$	[Chan: 2007]
Monge	$O(n^2)$	via [Aggarwal+: 1987]
implicit unit-Monge	$O(n^{1.5})$	[T: 2006]
	$O(n \log n)$	[T: 2010]

Matrix distance multiplication

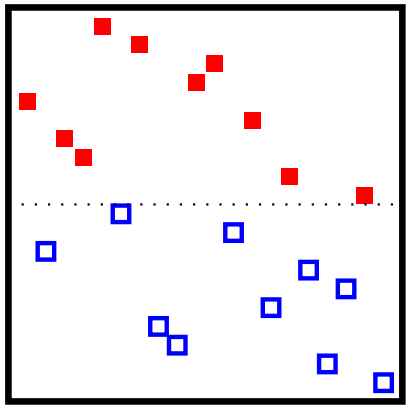
Seaweed matrix multiplication

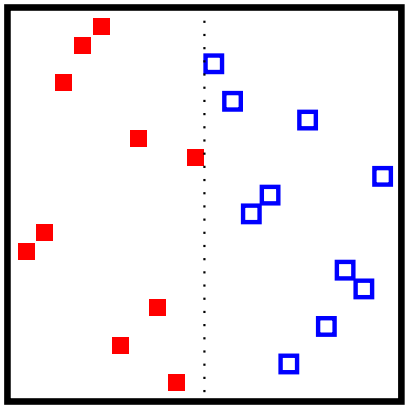


Matrix distance multiplication

Seaweed matrix multiplication

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



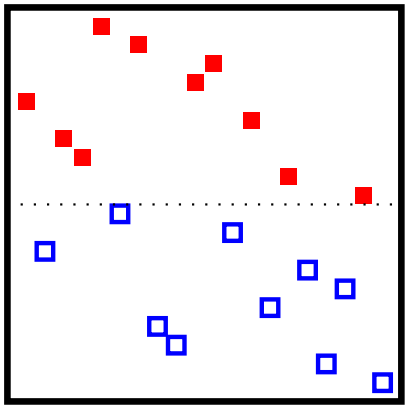


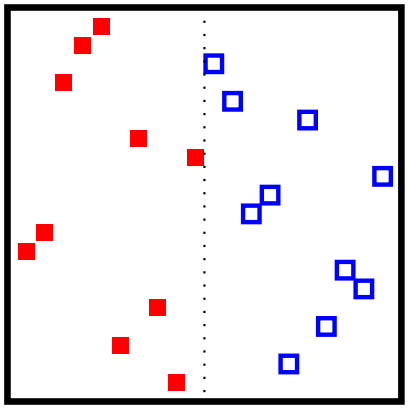
$P_{A,lo}$, $P_{A,hi}$

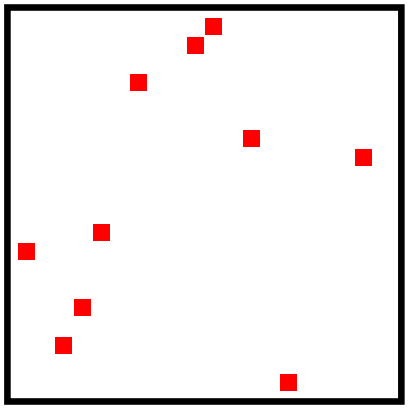
Matrix distance multiplication

Seaweed matrix multiplication

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





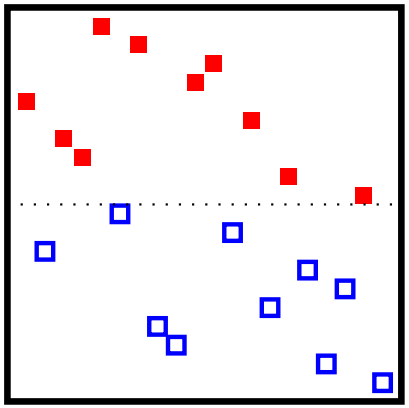


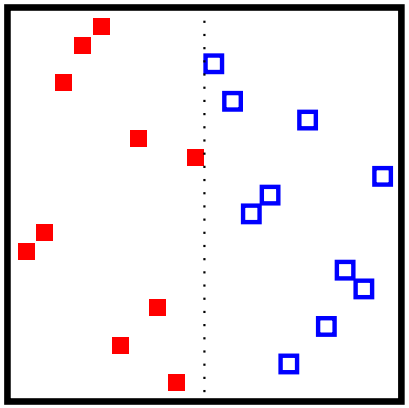
$P_{A,lo}$, $P_{A,hi}$

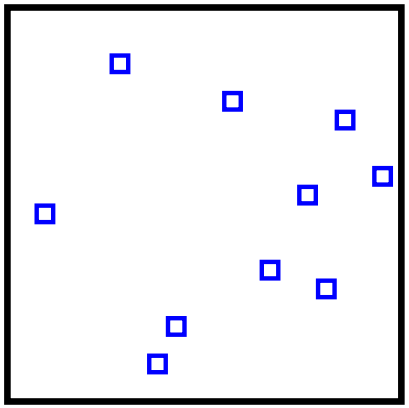
Matrix distance multiplication

Seaweed matrix multiplication

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





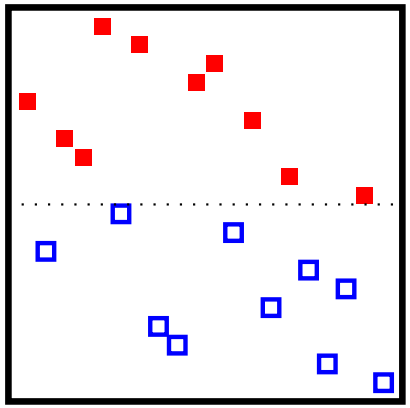


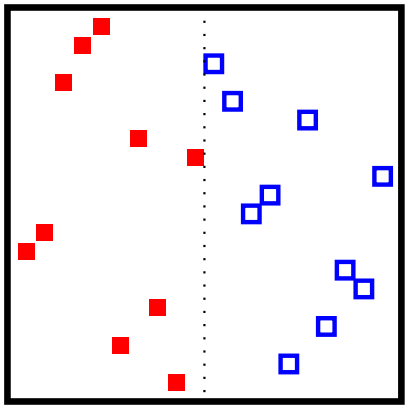
$P_{A,lo}$, $P_{A,hi}$

Matrix distance multiplication

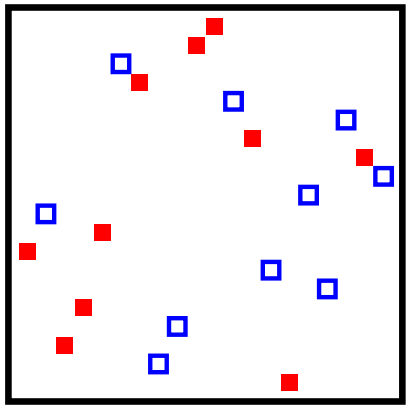
Seaweed matrix multiplication

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





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

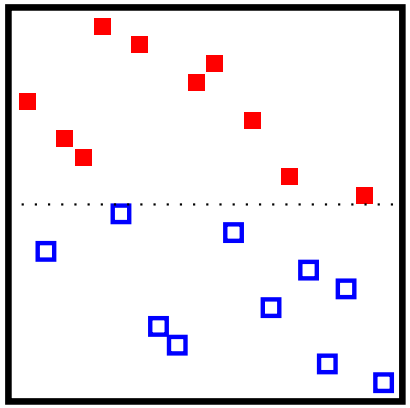


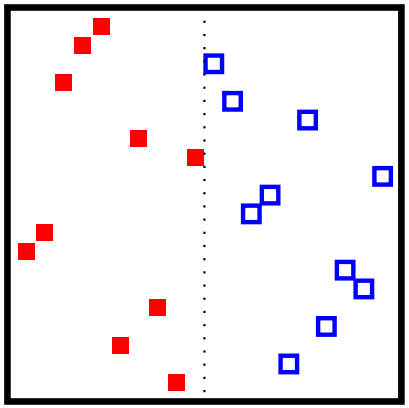
$P_{C,lo} + P_{C,hi}$

Matrix distance multiplication

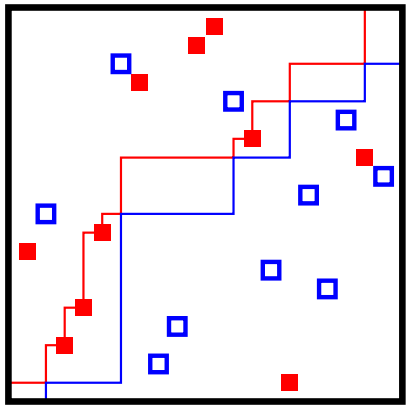
Seaweed matrix multiplication

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





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

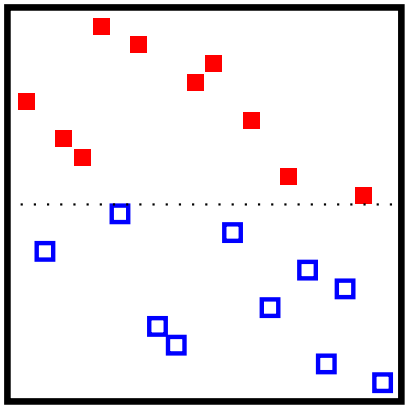


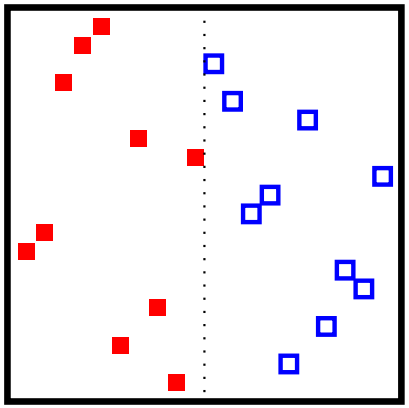
$P_{C,lo} + P_{C,hi}$

Matrix distance multiplication

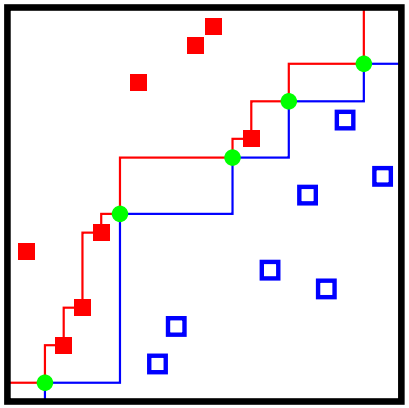
Seaweed matrix multiplication

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





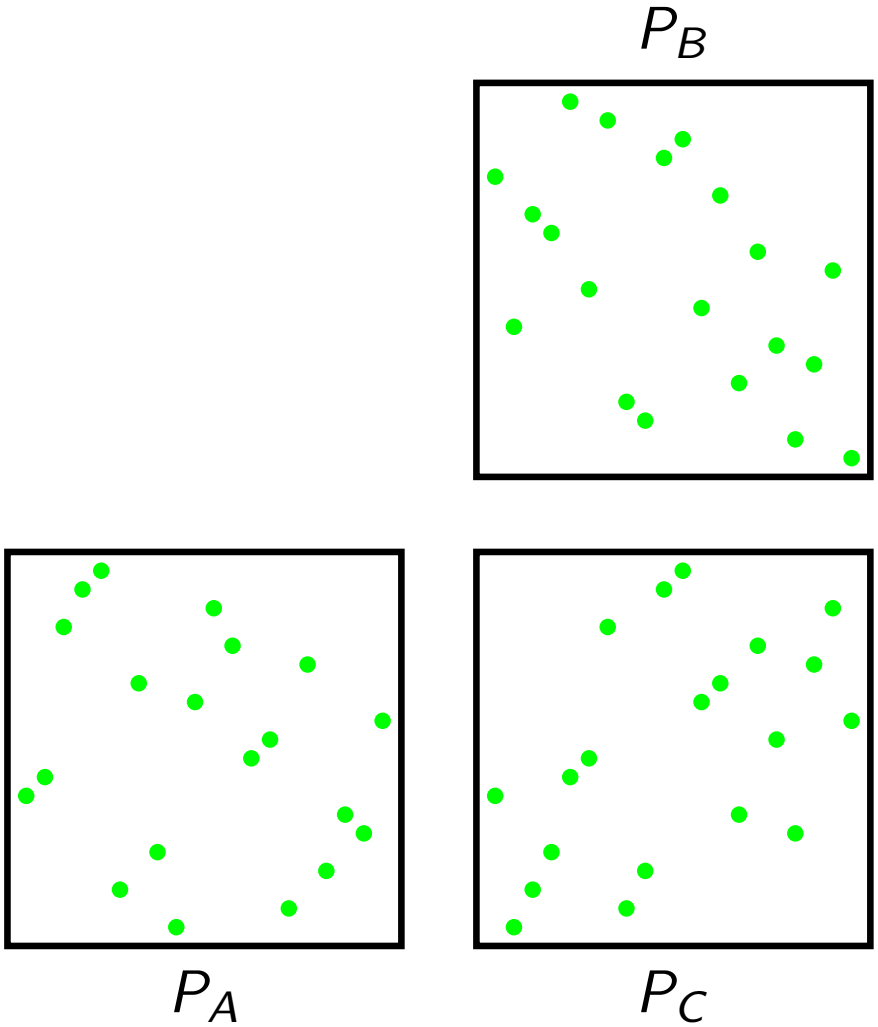
$P_{A,lo}$, $P_{A,hi}$



P_C

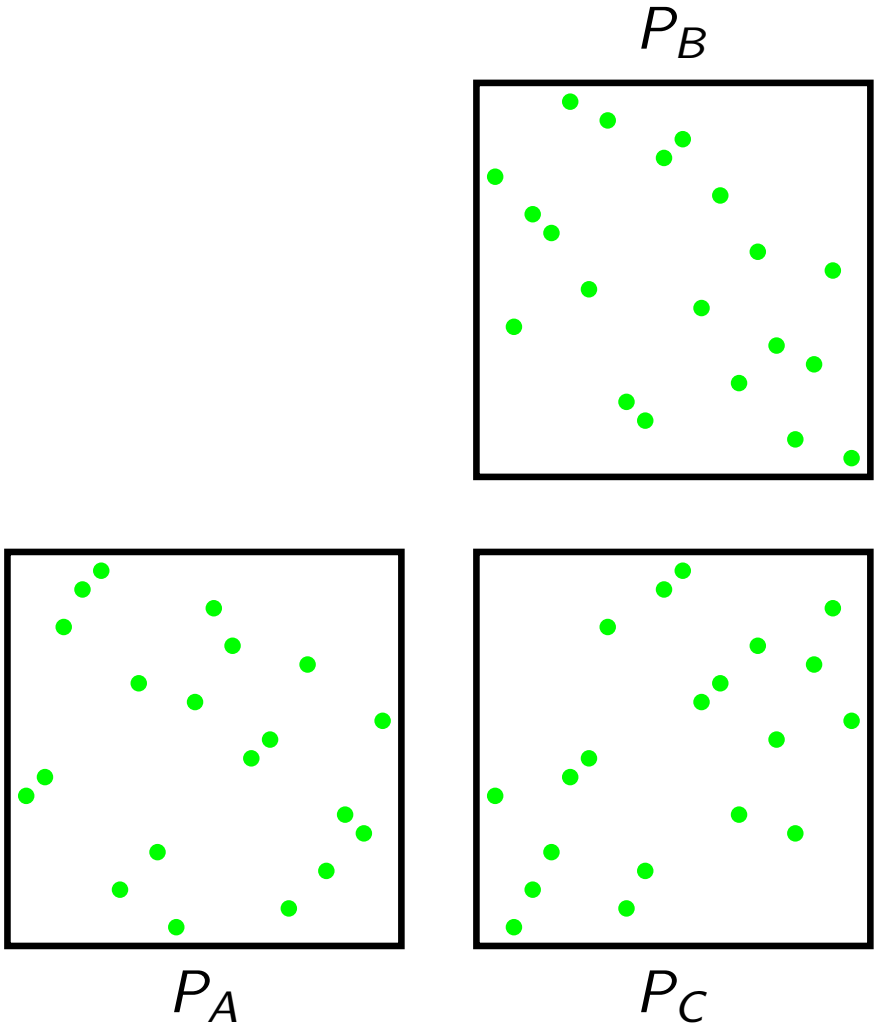
Matrix distance multiplication

Seaweed matrix multiplication



Matrix distance multiplication

Seaweed matrix multiplication



Implicit unit-Monge matrix $\odot$-multiplication: the algorithm

$P_C^\Sigma(i,k) = \min_j\big(P_A^\Sigma(i,j) + P_B^\Sigma(j,k)\big)$

Divide-and-conquer on the range of j: two subproblems of size $n/2$

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

Conquer: tracing a balanced path from bottom-left to top-right P_C

Path invariant: equal number of $\ll$-dominated nonzeros in $P_{C,hi}$ and $\ll$-dominating nonzeros in $P_{C,lo}$

All such nonzeros are "errors" in the output, must be removed

Must compensate by placing nonzeros on the path, time $O(n)$

Overall time $O(n \log n)$

Bruhat order

Permutation A is lower ("more sorted") than permutation B in the Bruhat order ($A \preceq B$), if $B \rightsquigarrow A$ by successive pairwise sorting (equivalently, $A \rightsquigarrow B$ by anti-sorting) of arbitrary pairs

Permutation matrices: $P_A \preceq P_B$, if $P_B \rightsquigarrow P_A$ by successive 2×2 submatrix sorting: $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right) \rightarrow \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$

Bruhat order

Permutation A is lower ("more sorted") than permutation B in the Bruhat order ($A \preceq B$), if $B \rightsquigarrow A$ by successive pairwise sorting (equivalently, $A \rightsquigarrow B$ by anti-sorting) of arbitrary pairs

Permutation matrices: $P_A \preceq P_B$, if $P_B \rightsquigarrow P_A$ by successive 2×2 submatrix sorting: $\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right) \to \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$

Plays an important role in group theory and algebraic geometry (inclusion order of Schubert varieties)

Describes pivoting order in Gaussian elimination (LUP/Bruhat decomposition)

Matrix distance multiplication

Bruhat order

Bruhat comparability: running time

$O(n^2)$	[Ehresmann: 1934; Proctor: 1982; Grigoriev: 1982]
$O(n \log n)$	[T: 2013]
$O\left(\frac{n \log n}{\log \log n}\right)$	[Gawrychowski: NEW]

Matrix distance multiplication

Bruhat order

Ehresmann's criterion (dot criterion, related to tableau criterion)

$P_A \preceq P_B$ iff $P_A^\Sigma \leq P_B^\Sigma$ elementwise

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}^\Sigma = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \leq \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}^\Sigma$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}^\Sigma = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \not\leq \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^\Sigma$$

Time $O(n^2)$

Matrix distance multiplication

Bruhat order

Seaweed criterion

$P_A \preceq P_B$ iff $P_A^R \boxdot P_B = Id^R$, where P^R = clockwise rotation of matrix P

Intuition: permutations represented by seaweed braids

$P_A \preceq P_B$, iff

- no pair of seaweeds is crossed in P_A, while the "corresponding" pair is uncrossed in P_B
- equivalently, no pair is uncrossed in P_A^R, while the "corresponding" pair is uncrossed in P_B
- equivalently, $P_A^R \boxdot P_B = Id^R$

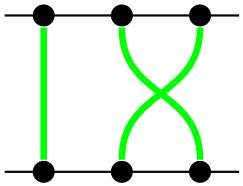
Time $O(n \log n)$ by seaweed matrix multiplication

Matrix distance multiplication

Bruhat order

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix} \preceq \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}$$

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}^{R} \boxdot \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix} \boxdot \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix} = \begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix} = Id^{R}$$

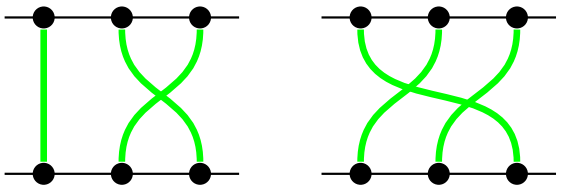
P_A 

Matrix distance multiplication

Bruhat order

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix} \preceq \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}$$

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix} \overset{R}{\boxdot} \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix} \boxdot \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix} = \begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix} = Id^R$$

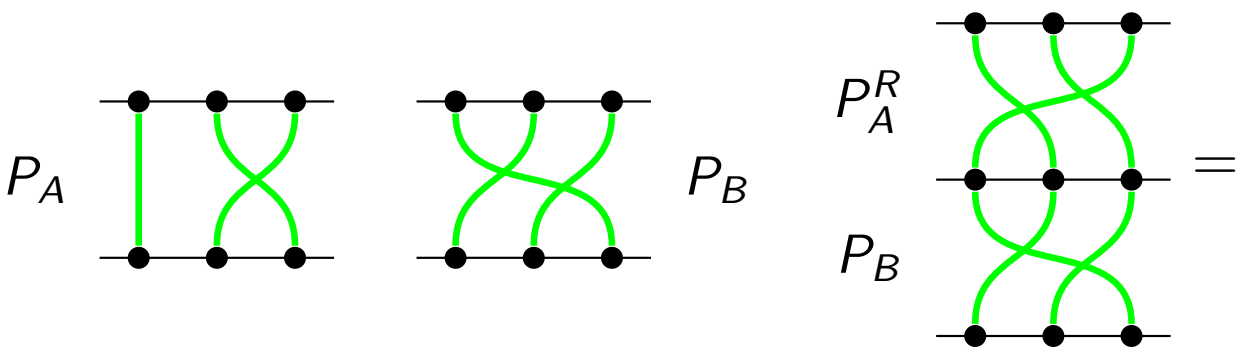
P_A  P_B

Matrix distance multiplication

Bruhat order

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix} \preceq \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}$$

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}^{R} \boxdot \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix} \boxdot \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix} = \begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix} = Id^{R}$$

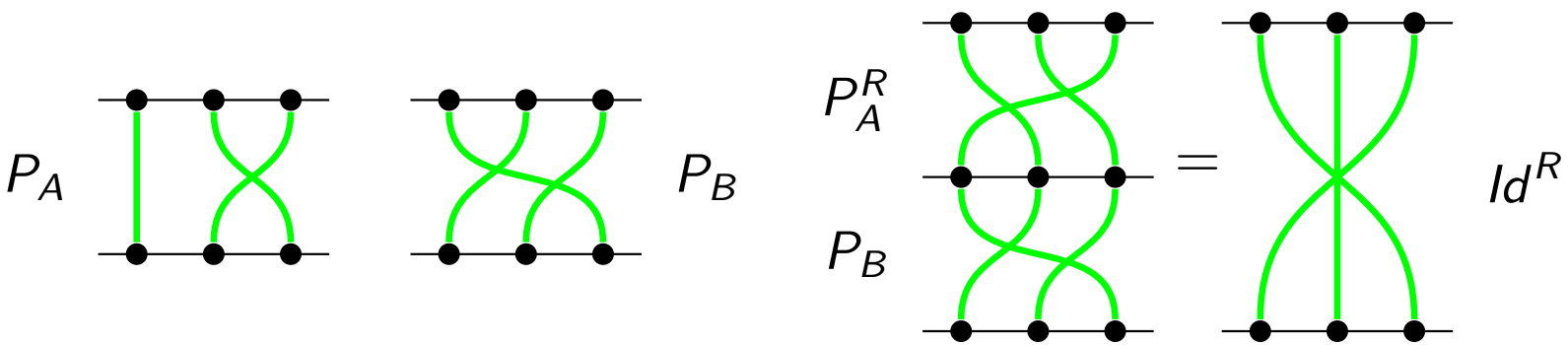


Matrix distance multiplication

Bruhat order

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix} \preceq \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}$$

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}^R \boxdot \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix} \boxdot \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix} = \begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix} = Id^R$$

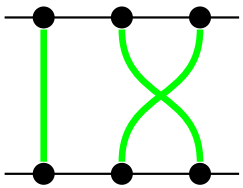


Matrix distance multiplication

Bruhat order

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix} \not\preceq \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix}$$

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}^{R} \boxdot \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix} \boxdot \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix} \neq Id^{R}$$

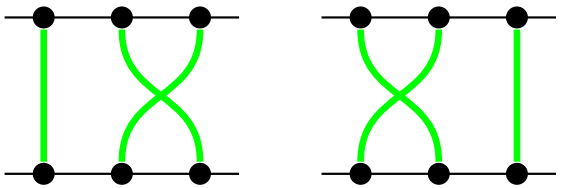
P_A 

Matrix distance multiplication

Bruhat order

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \not\preceq \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}^R \boxdot \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \boxdot \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \neq Id^R$$

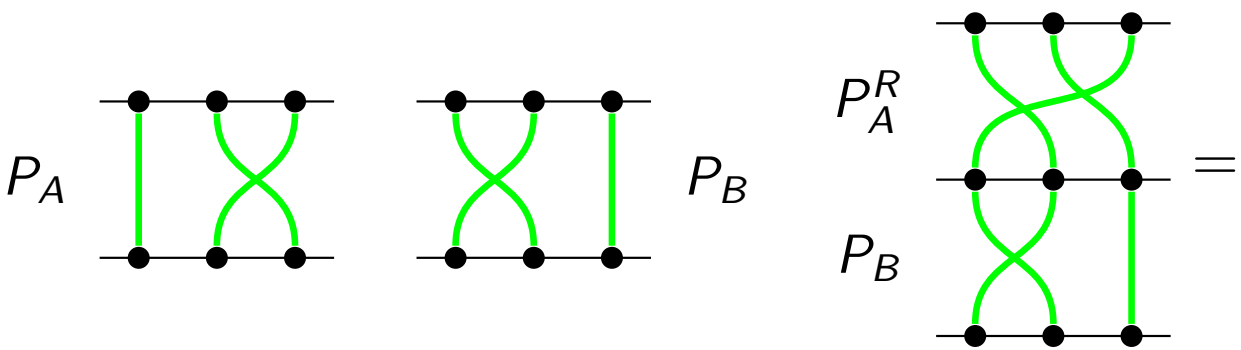
P_A  P_B

Matrix distance multiplication

Bruhat order

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix} \not\preceq \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix}$$

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}^R \boxdot \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix} \boxdot \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix} \neq Id^R$$

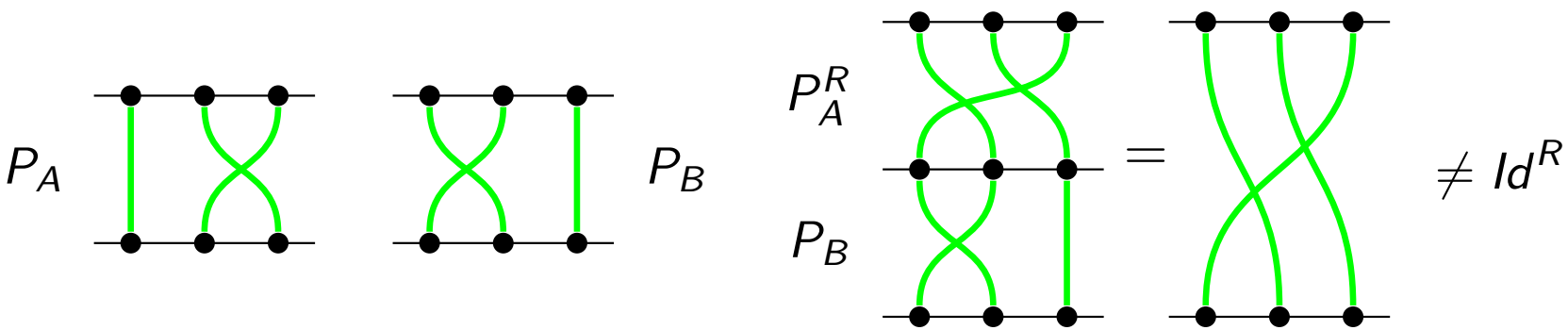


Matrix distance multiplication

Bruhat order

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix} \not\preceq \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix}$$

$$\begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}^{R} \boxdot \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix} \boxdot \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix} = \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix} \neq Id^R$$



Matrix distance multiplication

Bruhat order

Alternative solution: clever implementation of Ehresmann's criterion [Gawrychowski: 2013]

The online partial sums problem: maintain array $X[1:n]$, subject to

- $update(k, \Delta)$: $X[k] \leftarrow X[k] + \Delta$
- $prefixsum(k)$: return $\sum_{1 \leq i \leq k} X[i]$

Query time:

- $\Theta(\log n)$ in semigroup or group model
- $\Theta\left(\frac{\log n}{\log \log n}\right)$ in RAM model on integers [Pătraşcu, Demaine: 2004]

Gives Bruhat comparability in time $O\left(\frac{n \log n}{\log \log n}\right)$ in RAM model

Open problem: seaweed multiplication in time $O\left(\frac{n \log n}{\log \log n}\right)$?

Semi-local string comparison

Semi-local LCS and edit distance

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

Contiguous substrings vs 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

Semi-local string comparison

Semi-local LCS and edit distance

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

Contiguous substrings vs 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 $score(mismatch) = 0$

In biological terms, "loss-free alignment" (unlike efficient but "lossy" BLAST)

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]
$O\left(\frac{mn}{\log^2 n}\right)$	$\sigma = O(1)$	[Masek, Paterson: 1980]
		[Crochemore+: 2003]
$O\left(\frac{mn(\log\log n)^2}{\log^2 n}\right)$		[Paterson, Dančík: 1994]
		[Bille, Farach-Colton: 2008]

Running time varies depending on the RAM model version

We assume word-RAM with word size $\log n$ (where it matters)

LCS computation by dynamic programming

$$lcs(a, \emptyset) = 0$$
$$lcs(\emptyset, b) = 0$$

$$lcs(a\alpha, b\beta) = \begin{cases} \max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\ lcs(a, b) + 1 & \text{if } \alpha = \beta \end{cases}$$

LCS computation by dynamic programming

$$lcs(a, \emptyset) = 0$$
$$lcs(\emptyset, b) = 0$$

$$lcs(a\alpha, b\beta) = \begin{cases} \max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\ lcs(a, b) + 1 & \text{if } \alpha = \beta \end{cases}$$

	*	D	E	F	I	N	E
*	0	0	0	0	0	0	0
D	0						
E	0						
S	0						
I	0						
G	0						
N	0						

LCS computation by dynamic programming

$$lcs(a, \emptyset) = 0$$
$$lcs(\emptyset, b) = 0$$

$$lcs(a\alpha, b\beta) = \begin{cases} \max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\ lcs(a, b) + 1 & \text{if } \alpha = \beta \end{cases}$$

	*	D	E	F	I	N	E
*	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1
E	0						
S	0						
I	0						
G	0						
N	0						

LCS computation by dynamic programming

$$lcs(a, \emptyset) = 0$$
$$lcs(\emptyset, b) = 0$$

$$lcs(a\alpha, b\beta) = \begin{cases} \max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\ lcs(a, b) + 1 & \text{if } \alpha = \beta \end{cases}$$

	*	D	E	F	I	N	E
*	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1
E	0	1	2	2	2	2	2
S	0						
I	0						
G	0						
N	0						

LCS computation by dynamic programming

$$lcs(a, \emptyset) = 0$$
$$lcs(\emptyset, b) = 0$$

$$lcs(a\alpha, b\beta) = \begin{cases} \max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\ lcs(a, b) + 1 & \text{if } \alpha = \beta \end{cases}$$

	*	D	E	F	I	N	E
*	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1
E	0	1	2	2	2	2	2
S	0	1	2	2	2	2	2
I	0						
G	0						
N	0						

LCS computation by dynamic programming

$$lcs(a, \emptyset) = 0$$
$$lcs(\emptyset, b) = 0$$

$$lcs(a\alpha, b\beta) = \begin{cases} \max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\ lcs(a, b) + 1 & \text{if } \alpha = \beta \end{cases}$$

	*	D	E	F	I	N	E
*	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1
E	0	1	2	2	2	2	2
S	0	1	2	2	2	2	2
I	0	1	2	2	3	3	3
G	0						
N	0						

LCS computation by dynamic programming

$$lcs(a, \emptyset) = 0$$
$$lcs(\emptyset, b) = 0$$

$$lcs(a\alpha, b\beta) = \begin{cases} \max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\ lcs(a, b) + 1 & \text{if } \alpha = \beta \end{cases}$$

	*	D	E	F	I	N	E
*	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1
E	0	1	2	2	2	2	2
S	0	1	2	2	2	2	2
I	0	1	2	2	3	3	3
G	0	1	2	2	3	3	3
N	0	1	2	2	3	4	4

LCS computation by dynamic programming

$$lcs(a, \emptyset) = 0$$
$$lcs(\emptyset, b) = 0$$

$$lcs(a\alpha, b\beta) = \begin{cases} \max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\ lcs(a, b) + 1 & \text{if } \alpha = \beta \end{cases}$$

	*	D	E	F	I	N	E
*	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1
E	0	1	2	2	2	2	2
S	0	1	2	2	2	2	2
I	0	1	2	2	3	3	3
G	0	1	2	2	3	3	3
N	0	1	2	2	3	4	4

$lcs(\text{“DEFINE”}, \text{“DESIGN”}) = 4$

LCS computation by dynamic programming

$$lcs(a, \emptyset) = 0$$
$$lcs(\emptyset, b) = 0$$

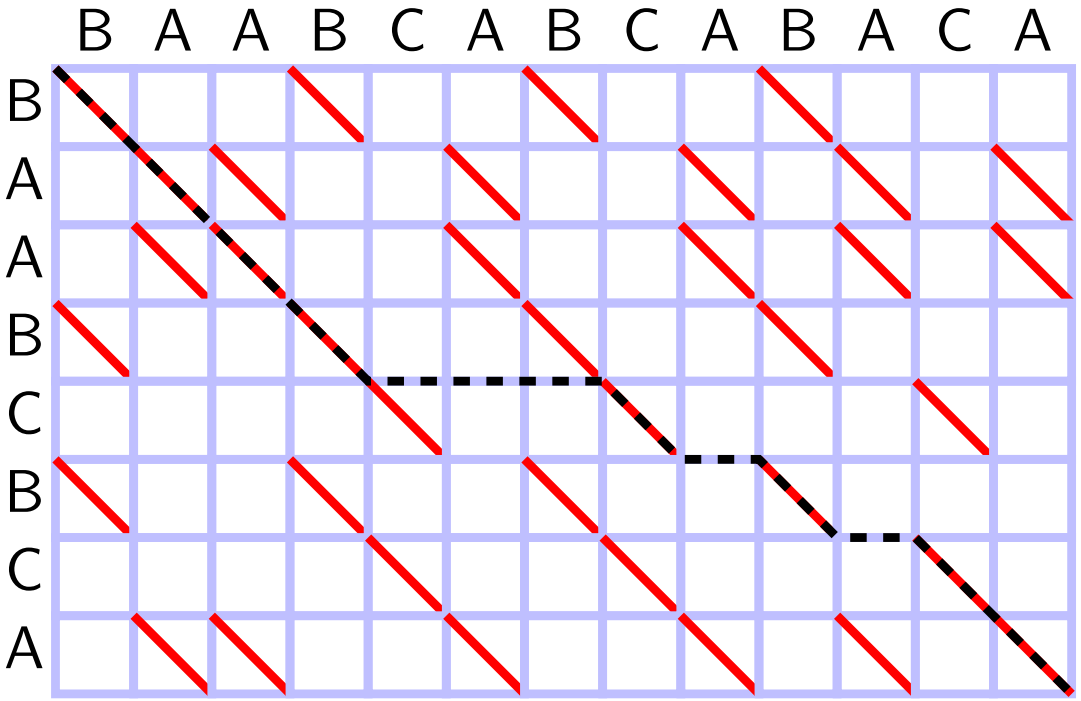
$$lcs(a\alpha, b\beta) = \begin{cases} \max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\ lcs(a, b) + 1 & \text{if } \alpha = \beta \end{cases}$$

	*	D	E	F	I	N	E
*	0	0	0	0	0	0	0
D	0	1	1	1	1	1	1
E	0	1	2	2	2	2	2
S	0	1	2	2	2	2	2
I	0	1	2	2	3	3	3
G	0	1	2	2	3	3	3
N	0	1	2	2	3	4	4

$lcs(\text{“DEFINE”}, \text{“DESIGN”}) = 4$

$LCS(a, b)$ can be traced back through the dynamic programming table at no extra asymptotic time cost

LCS on the alignment graph (directed, acyclic)



$blue = 0$

$red = 1$

$score(\text{“BAABCBCA”}, \text{“BAABCABCABACA”}) = len(\text{“BAABCBCA”}) = 8$

LCS = highest-score path from top-left to bottom-right

LCS: dynamic programming [WF: 1974]

Sweep cells in any $\ll$-compatible order

Cell update: time $O(1)$

Overall time $O(mn)$

LCS: micro-block dynamic programming [MP: 1980; BF: 2008]

Sweep cells in micro-blocks, in any $\ll$-compatible order

Micro-block size:

- $t = O(\log n)$ when $\sigma = O(1)$
- $t = O\left(\frac{\log n}{\log \log n}\right)$ otherwise

Micro-block interface:

- $O(t)$ characters, each $O(\log \sigma)$ bits, can be reduced to $O(\log t)$ bits
- $O(t)$ small integers, each $O(1)$ bits

Micro-block update: time $O(1)$, by precomputing all possible interfaces

Overall time $O\left(\frac{mn}{\log^2 n}\right)$ when $\sigma = O(1)$, $O\left(\frac{mn(\log \log n)^2}{\log^2 n}\right)$ otherwise

'Begin at the beginning,' the King said gravely, 'and go on till you come to the end: then stop.'

L. Carroll, *Alice in Wonderland*

'Begin at the beginning,' the King said gravely, 'and go on till you come to the end: then stop.'

L. Carroll, *Alice in Wonderland*

Dynamic programming: begins at empty strings, proceeds by appending characters, then stops

What about:

- prepending/deleting characters (dynamic LCS)
- concatenating strings (LCS on compressed strings; parallel LCS)
- taking substrings (= local alignment)

Semi-local string comparison

Semi-local LCS and edit distance

Dynamic programming from both ends: better by $\times 2$, but still not good enough

Is dynamic programming strictly necessary to solve sequence alignment problems?

Eppstein+, *Efficient algorithms for sequence analysis*, 1991

The semi-local LCS problem

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

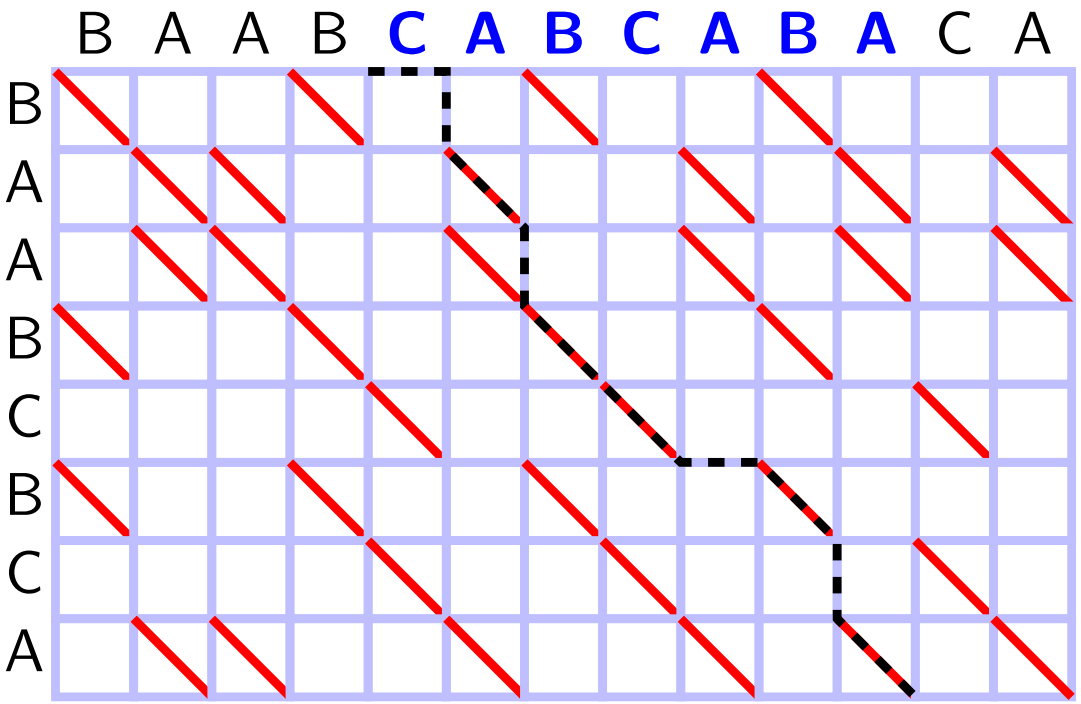
- string-substring LCS: string a vs every substring of b
- prefix-suffix LCS: every prefix of a vs every suffix of b
- suffix-prefix LCS: every suffix of a vs every prefix of b
- substring-string LCS: every substring of a vs string b

The semi-local LCS problem

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

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

Semi-local LCS on the alignment graph



$blue = 0$

$red = 1$

$score(\text{“BAABCBCA”}, \text{“}\mathbf{CABCABA}\text{”}) = len(\text{“ABCBA”}) = 5$

String-substring LCS: all highest-score top-to-bottom paths

Semi-local LCS: all highest-score boundary-to-boundary paths

The score matrix H

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"

$b =$ "BAAB**CABCABA**CA"

$H(i,j) = score(a, b\langle i : j\rangle)$

$H(4,11) = 5$

$H(i,j) = j - i$ 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]
...or any 2D orthogonal range counting data structure			

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

Semi-local string comparison

Score matrices and seaweed matrices

The score matrix H and the seaweed matrix P

$H(i,j)$: the number of matched characters for a vs substring $b\langle i:j\rangle$

$j-i-H(i,j)$: the number of unmatched characters

Properties of matrix $j-i-H(i,j)$:

- simple unit-Monge
- therefore, $= P^{\Sigma}$, where $P = -H^{\square}$ is a permutation matrix

P is the seaweed matrix, giving an implicit representation of H

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

Semi-local string comparison

Score matrices and seaweed matrices

The score matrix H and the seaweed matrix P

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"

$b =$ "BAAB**CABCABA**CA"

$H(i,j) = score(a, b\langle i : j\rangle)$

$H(4,11) = 5$

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

Semi-local string comparison

Score matrices and seaweed matrices

The score matrix H and the seaweed matrix P

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"

$b =$ "BAAB**CABCABA**CA"

$H(i,j) = score(a, b\langle i : j\rangle)$

$H(4,11) = 5$

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

blue: difference in H is 0

red: difference in H is 1

Semi-local string comparison

Score matrices and seaweed matrices

The score matrix H and the seaweed matrix P

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"

$b =$ "BAAB**CABCABA**CA"

$H(i,j) = score(a, b\langle i : j\rangle)$

$H(4,11) = 5$

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

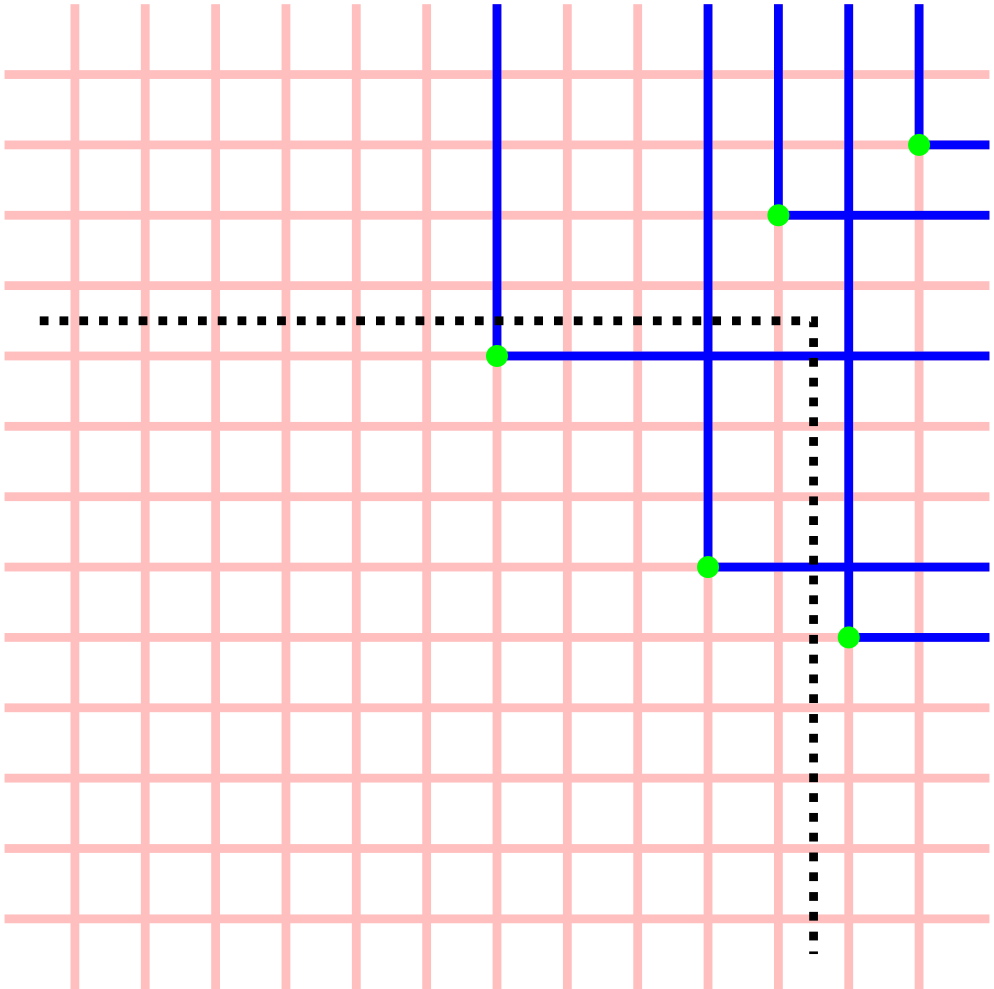
blue: difference in H is 0

red: difference in H is 1

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

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

The score matrix H and the seaweed matrix P



$a =$ "BAABCBCA"

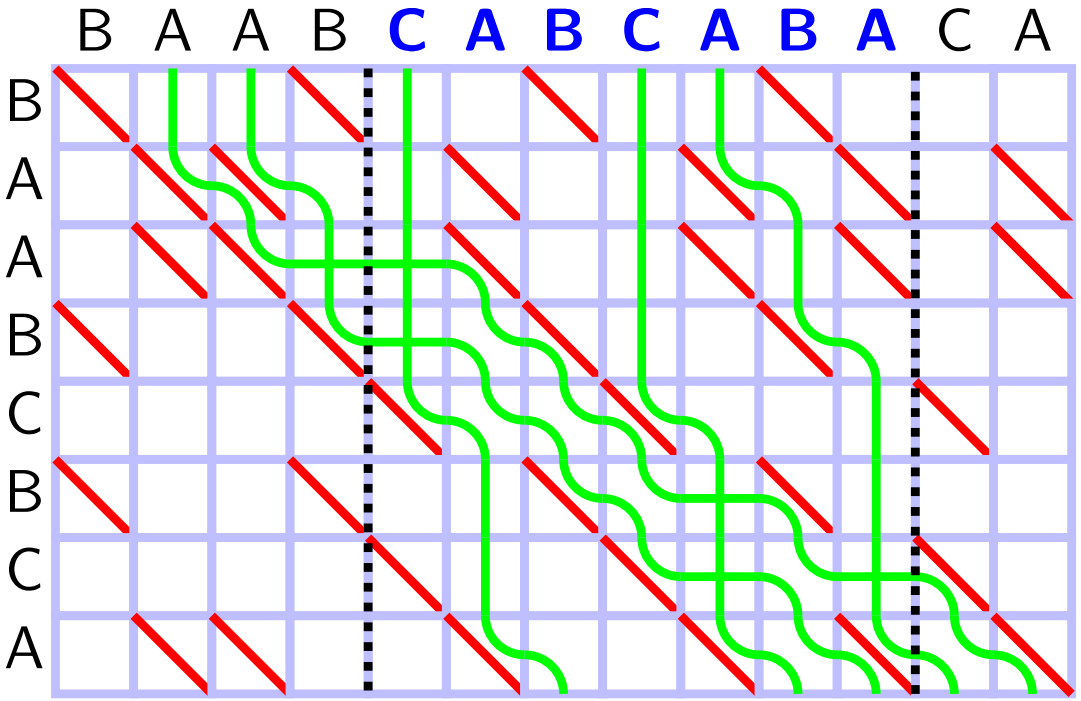
$b =$ "BAAB**CABCABA**CA"

$H(4, 11) =$
$11 - 4 - P^{\Sigma}(i, j) =$
$11 - 4 - 2 = 5$

Semi-local string comparison

Score matrices and seaweed matrices

The (combed) seaweed braid in the alignment graph



$a =$ "BAABCBCA"

$b =$ "BAAB**CABCABA**CA"

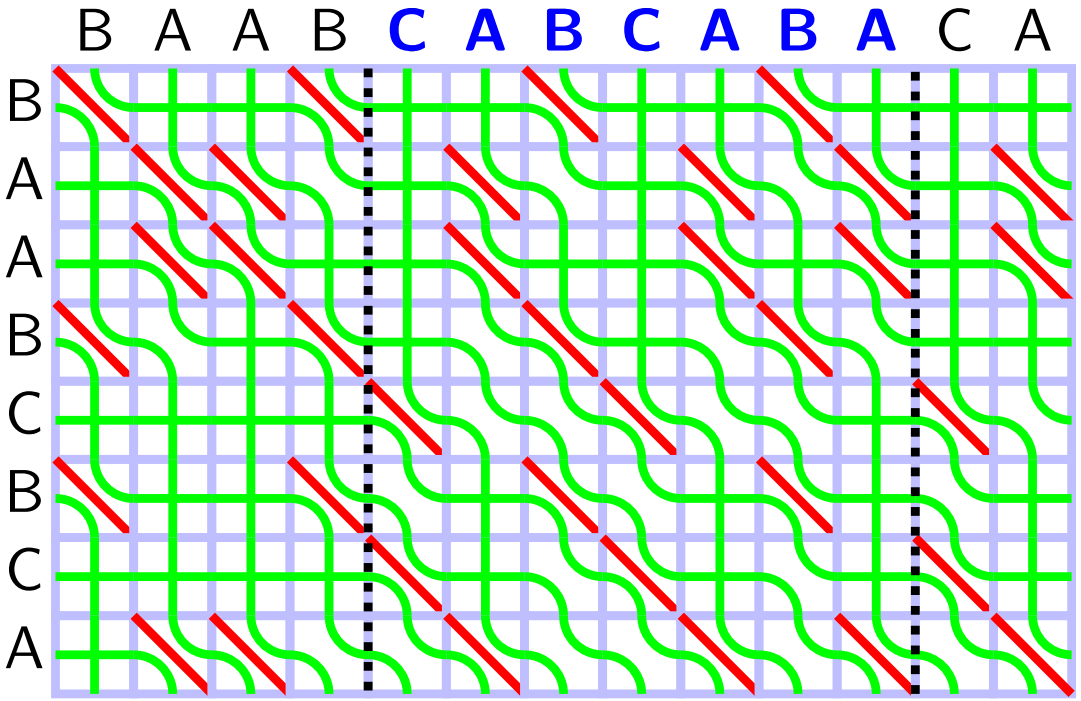
$H(4, 11) =$
$11 - 4 - P^{\Sigma}(i, j) =$
$11 - 4 - 2 = 5$

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

Semi-local string comparison

Score matrices and seaweed matrices

The (combed) seaweed braid in the alignment graph



$a =$ "BAABCBCA"

$b =$ "BAAB**CABCABA**CA"

$H(4, 11) =$
$11 - 4 - P^{\Sigma}(i, j) =$
$11 - 4 - 2 = 5$

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

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

Represent implicitly semi-local LCS for each prefix of a vs b

Semi-local string comparison

Score matrices and seaweed matrices

Seaweed braid: a highly symmetric object (element of $H_0(S_n)$)

Can be built by assembling subbraids: divide-and-conquer

Flexible approach to local alignment, compressed approximate matching, parallel computation...

Semi-local string comparison

Weighted alignment

The LCS problem is a special case of the weighted alignment score problem with weighted matches (w_M), mismatches (w_X) and gaps (w_G)

- LCS score: $w_M = 1,\ w_X = w_G = 0$
- Levenshtein score: $w_M = 2,\ w_X = 1,\ w_G = 0$

Semi-local string comparison

Weighted alignment

The LCS problem is a special case of the weighted alignment score problem with weighted matches (w_M), mismatches (w_X) and gaps (w_G)

- LCS score: $w_M = 1,\ w_X = w_G = 0$
- Levenshtein score: $w_M = 2,\ w_X = 1,\ w_G = 0$

Alignment score is rational, if $w_M,\ w_X,\ w_G$ are rational numbers

Semi-local string comparison

Weighted alignment

The LCS problem is a special case of the weighted alignment score problem with weighted matches (w_M), mismatches (w_X) and gaps (w_G)

- LCS score: $w_M = 1,\ w_X = w_G = 0$
- Levenshtein score: $w_M = 2,\ w_X = 1,\ w_G = 0$

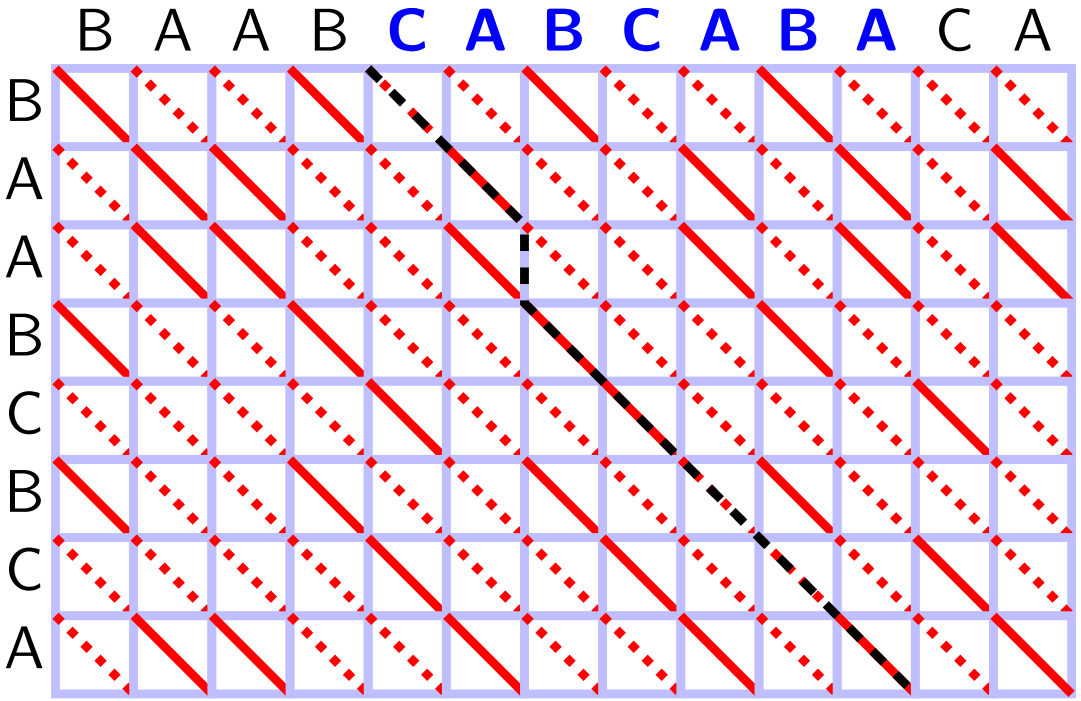
Alignment score is rational, if $w_M,\ w_X,\ w_G$ are rational numbers

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

Corresponds to weighted alignment score with $w_M = 0$:

- insertion/deletion cost $-w_G$
- substitution weight $-w_X$

Weighted alignment graph for a, b



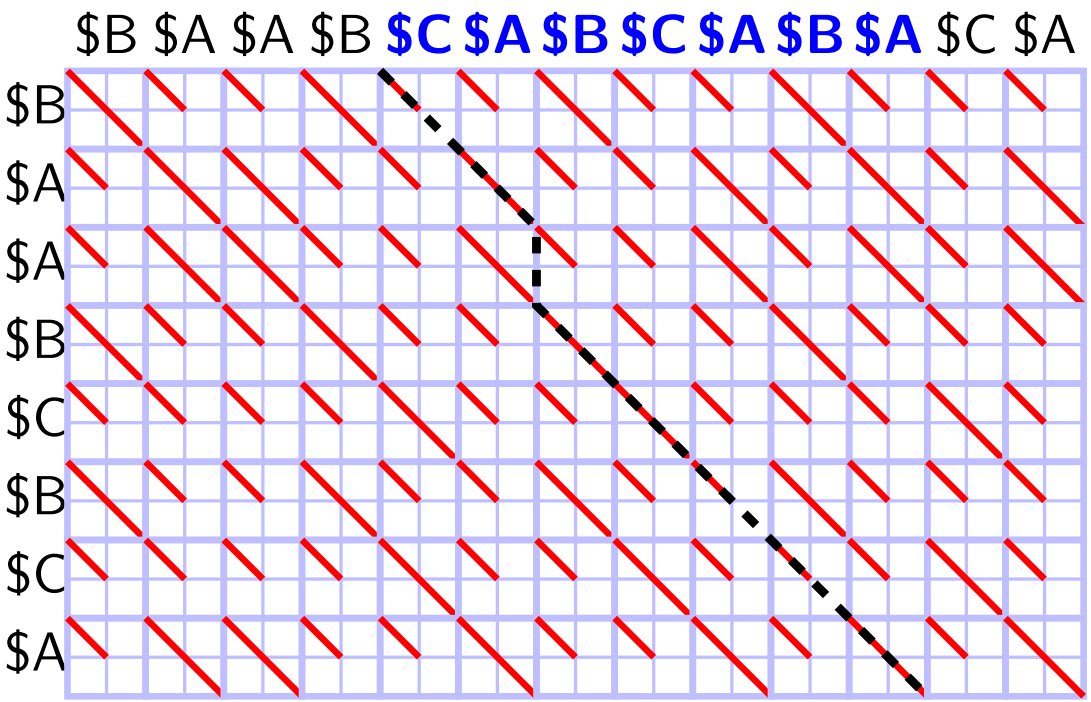
$blue = 0$
$red\ (solid) = 2$
$red\ (dotted) = 1$

$Levenshtein($"BAABCBCA", "**CABCABA**"$) = 11$

Semi-local string comparison

Weighted alignment

Reduction: ordinary alignment graph for blown-up a, b



$blue = 0$

$red = 1$ or 2

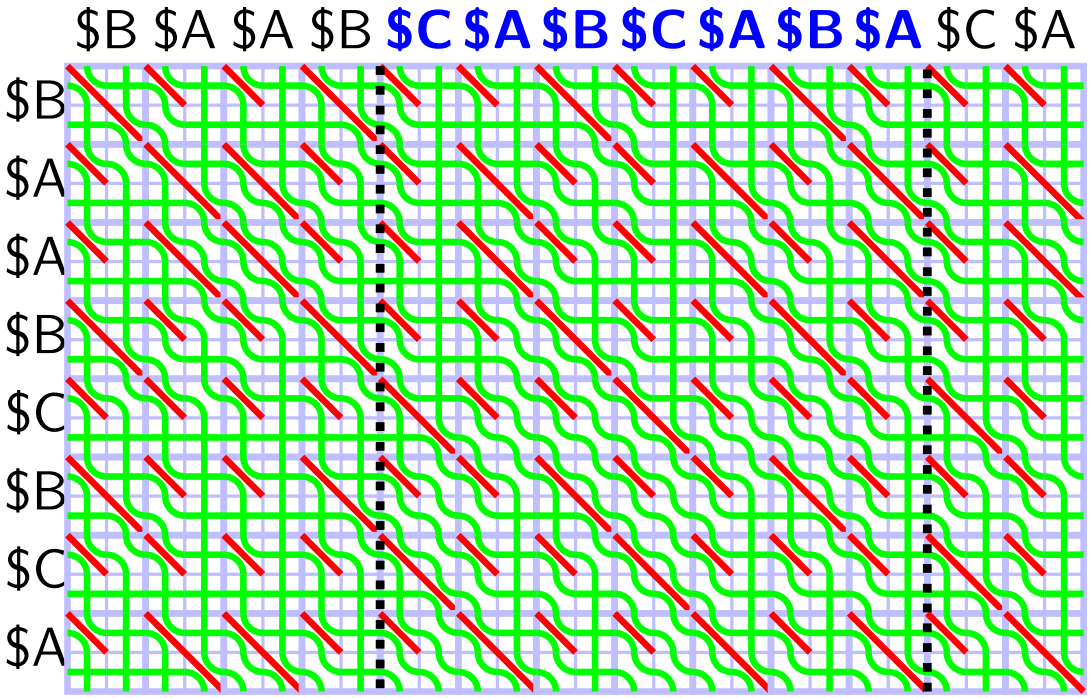
$Levenshtein($"BAABCBCA", "**CABCABA**"$) =$

$lcs($"BAABCBCA", "**CABCAB$A**"$) = 11$

Semi-local string comparison

Weighted alignment

Reduction: rational-weighted semi-local alignment to semi-local LCS



Let $w_M = 1$, $w_X = \frac{\mu}{\nu}$, $w_G = 0$

Increase $\times \nu^2$ in complexity (can be reduced to ν)

Compressed string comparison

Grammar compression

Notation: pattern p of length m; text t of length n

A GC-string (grammar-compressed string) t is a straight-line program (context-free grammar) generating $t = t_{\bar{n}}$ by $\bar{n}$ assignments of the form

- $t_k = \alpha$, where α is an alphabet character
- $t_k = uv$, where each of u, v is an alphabet character, or t_i for $i < k$

In general, $n = O(2^{\bar{n}})$

Example: Fibonacci string "ABAABABAABAAB"

$t_1 = \mathsf{A} \quad t_2 = t_1\mathsf{B} \quad t_3 = t_2t_1 \quad t_4 = t_3t_2 \quad t_5 = t_4t_3 \quad t_6 = t_5t_4$

Compressed string comparison

Grammar compression

Grammar-compression covers various compression types, e.g. LZ78, LZW (not LZ77 directly)

Simplifying assumption: arithmetic up to n runs in $O(1)$

This assumption can be removed by careful index remapping

Compressed string comparison

Extended substring-string LCS on GC-strings

LCS: running time ($r = m + n$, $\bar{r} = \bar{m} + \bar{n}$)

p	t			
plain	plain	$O(mn)$		[Wagner, Fischer: 1974]
		$O\left(\frac{mn}{\log^2 m}\right)$		[Masek, Paterson: 1980]
				[Crochemore+: 2003]
plain	GC	$O(m^3\bar{n} + \ldots)$	gen. CFG	[Myers: 1995]
		$O(m^{1.5}\bar{n})$	ext subs-s	[T: 2008]
		$O(m \log m \cdot \bar{n})$	ext subs-s	[T: 2010]
GC	GC	NP-hard		[Lifshits: 2005]
		$O(r^{1.2}\bar{r}^{1.4})$	R weights	[Hermelin+: 2009]
		$O(r \log r \cdot \bar{r})$		[T: 2010]
		$O\left(r \log(r/\bar{r}) \cdot \bar{r}\right)$		[Hermelin+: 2010]
		$O\left(r \log^{1/2}(r/\bar{r}) \cdot \bar{r}\right)$		[Gawrychowski: 2012]

Extended substring-string LCS (plain pattern, GC text): the algorithm

For every k, compute by recursion the appropriate part of seaweed matrix P_{p,t_k}, using matrix $\boxdot$-multiplication: time $O(m \log m \cdot \bar{n})$

Overall time $O(m \log m \cdot \bar{n})$

The global subsequence recognition problem

Does pattern p appear in text t as a subsequence?

Global subsequence recognition: running time

p	t		
plain	plain	$O(n)$	greedy
plain	GC	$O(m\bar{n})$	greedy
GC	GC	NP-hard	[Lifshits: 2005]

The local subsequence recognition problem

Find all minimally matching substrings of t with respect to p

Substring of t is matching, if p is a subsequence of t

Matching substring of t is minimally matching, if none of its proper substrings are matching

Compressed string comparison

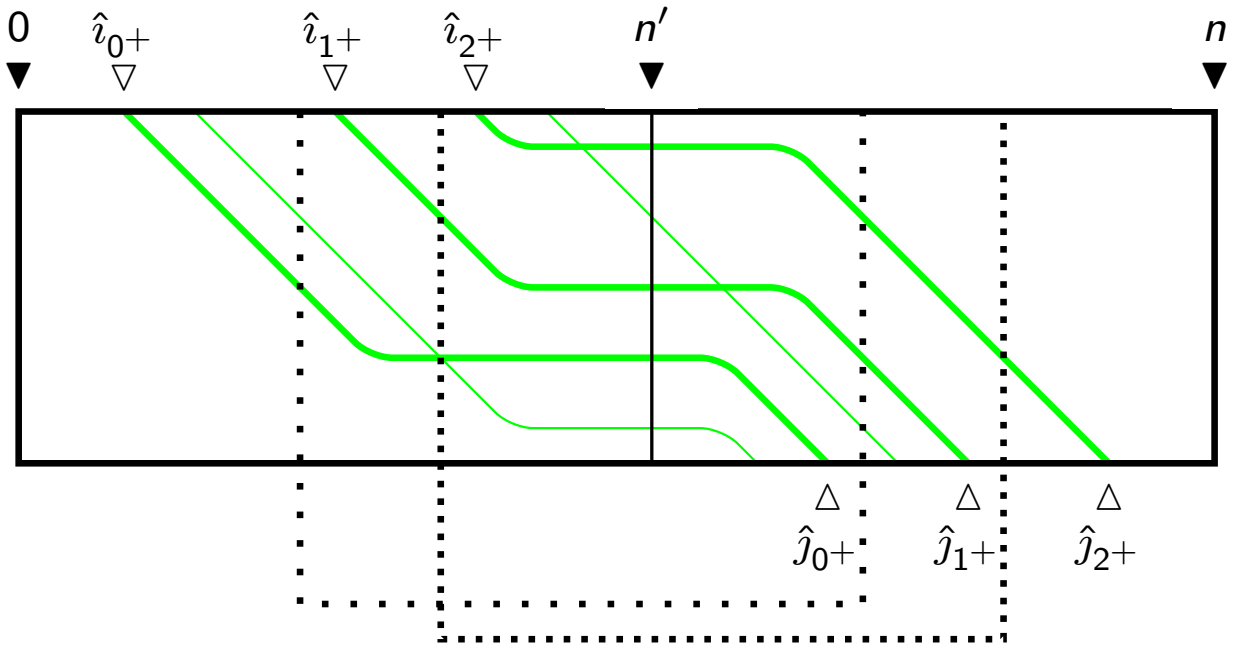
Subsequence recognition on GC-strings

Local subsequence recognition: running time (+ *output*)

p	t		
plain	plain	$O(mn)$	[Mannila+: 1995]
		$O(\frac{mn}{\log m})$	[Das+: 1997]
		$O(c^m + n)$	[Boasson+: 2001]
		$O(m + n\sigma)$	[Troniček: 2001]
plain	GC	$O(m^2 \log m\bar{n})$	[Cégielski+: 2006]
		$O(m^{1.5}\bar{n})$	[T: 2008]
		$O(m \log m \cdot \bar{n})$	[T: 2010]
		$O(m \cdot \bar{n})$	[Yamamoto+: 2011]
GC	GC	NP-hard	[Lifshits: 2005]

Compressed string comparison

Subsequence recognition on GC-strings



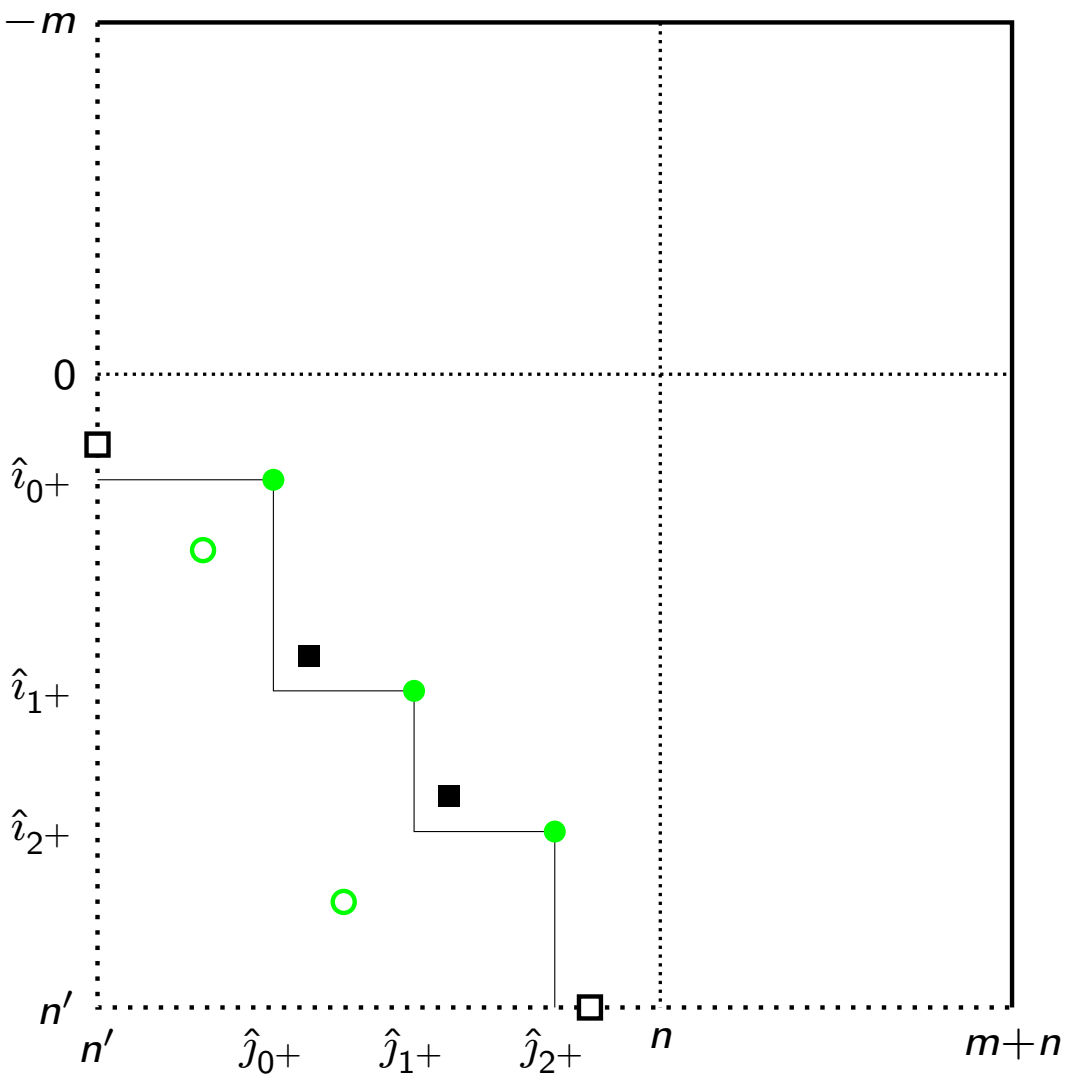
$b\langle i : j\rangle$ matching iff box $[i : j]$ not pierced left-to-right

Determined by $\ll$-chain of $\gtrsim$-maximal seaweeds

$b\langle i : j\rangle$ minimally matching iff (i, j) is in the interleaved skeleton $\ll$-chain

Compressed string comparison

Subsequence recognition on GC-strings



Local subsequence recognition (plain pattern, GC text): the algorithm

For every k, compute by recursion the appropriate part of seaweed matrix P_{p,t_k}, using matrix $\boxdot$-multiplication: time $O(m \log m \cdot \bar{n})$

Local subsequence recognition (plain pattern, GC text): the algorithm

For every k, compute by recursion the appropriate part of seaweed matrix P_{p,t_k}, using matrix $\boxdot$-multiplication: time $O(m \log m \cdot \bar{n})$

Given an assignment $t = t't''$, find by recursion

- minimally matching substrings in t'
- minimally matching substrings in t''

Local subsequence recognition (plain pattern, GC text): the algorithm

For every k, compute by recursion the appropriate part of seaweed matrix P_{p,t_k}, using matrix $\boxdot$-multiplication: time $O(m \log m \cdot \bar{n})$

Given an assignment $t = t't''$, find by recursion

- minimally matching substrings in t'
- minimally matching substrings in t''

Then, find $\ll$-chain of $\gtrdot$-maximal seaweeds in time $\bar{n} \cdot O(m) = O(m\bar{n})$

Its skeleton $\ll$-chain: minimally matching substrings in t overlapping t', t''

Overall time $O(m \log m \cdot \bar{n}) + O(m\bar{n}) = O(m \log m \cdot \bar{n})$

The threshold approximate matching problem

Find all matching substrings of t with respect to p, according to a threshold k

Substring of t is matching, if the edit distance for p vs t is at most k

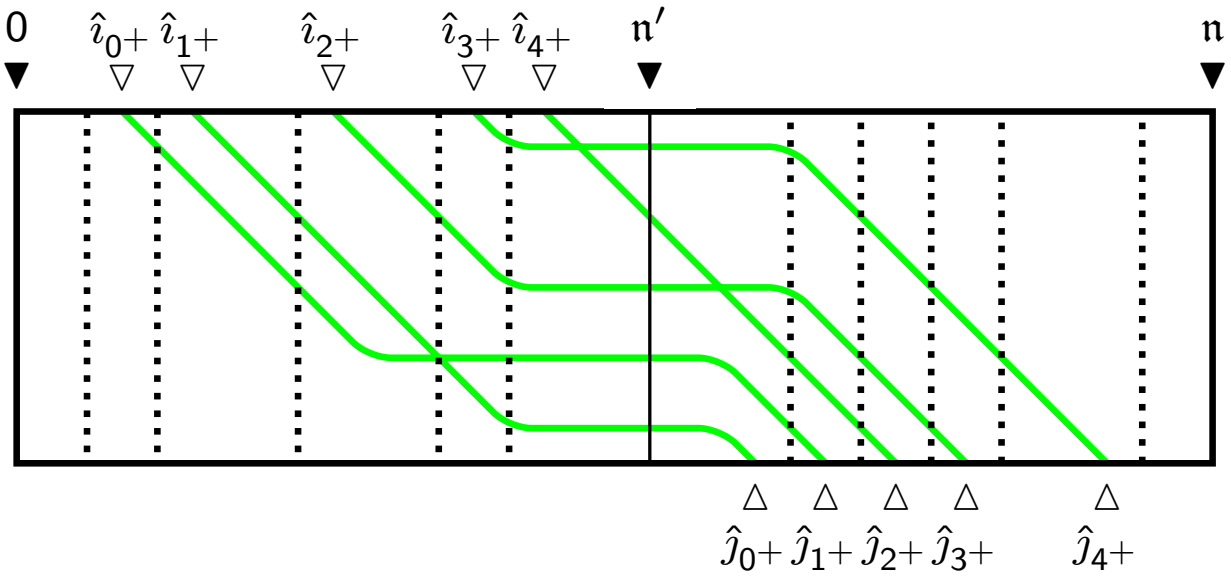
Threshold approximate matching: running time (+ *output*)

p	t		
plain	plain	$O(mn)$	[Sellers: 1980]
		$O(mk)$	[Landau, Vishkin: 1989]
		$O(m + n + \frac{nk^4}{m})$	[Cole, Hariharan: 2002]
plain	GC	$O(m\bar{n}k^2)$	[Kärkkäinen+: 2003]
		$O(m\bar{n}k + \bar{n}\log n)$	[LV: 1989] via [Bille+: 2010]
		$O(m\bar{n} + \bar{n}k^4 + \bar{n}\log n)$	[CH: 2002] via [Bille+: 2010]
		$O(m\log m \cdot \bar{n})$	[T: NEW]
GC	GC	NP-hard	[Lifshits: 2005]

(Also many specialised variants for LZ compression)

Compressed string comparison

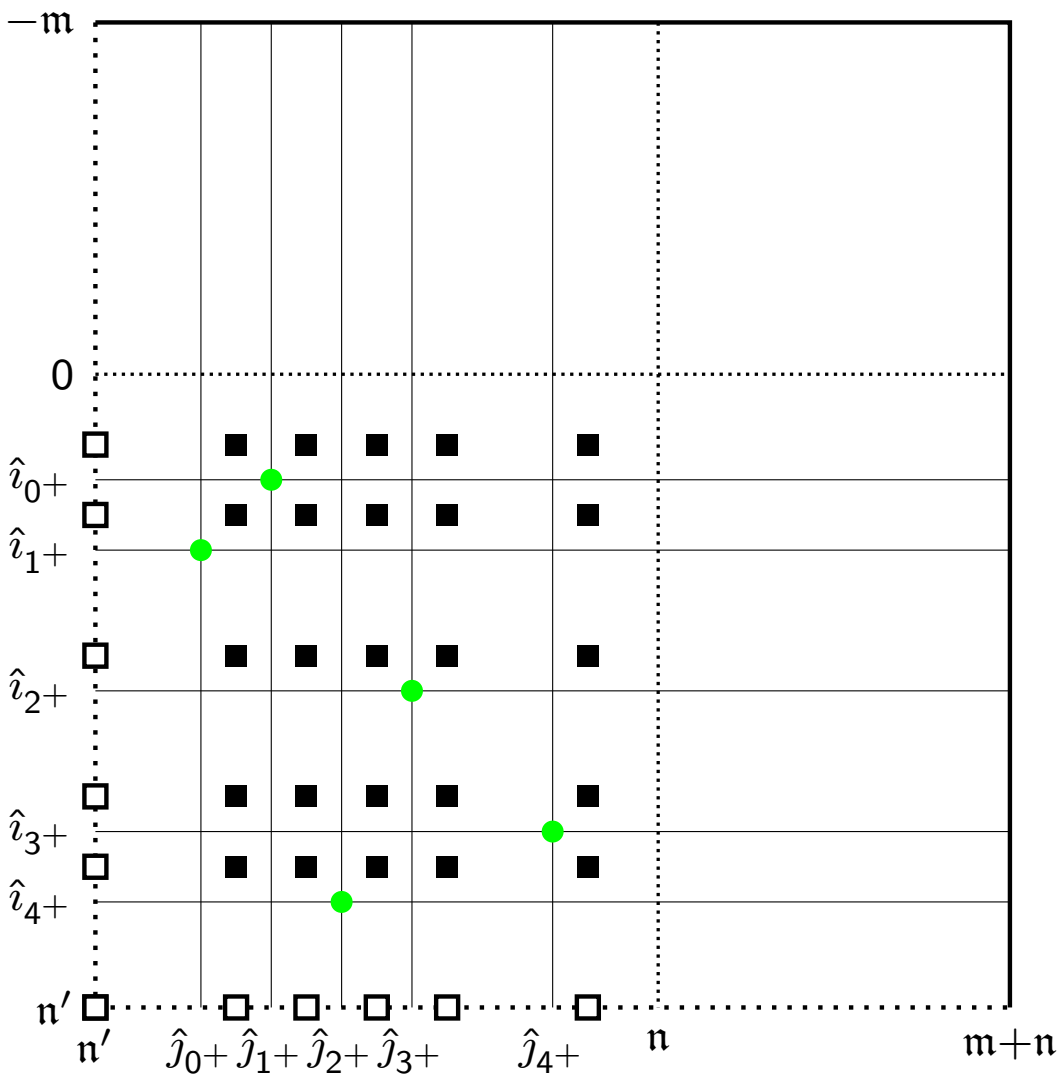
Threshold approximate matching



Blow up: weighted alignment on strings p, t of size m, n equivalent to LCS on strings $\mathfrak{p}$, $\mathfrak{t}$ of size $\mathfrak{m} = \nu m$, $\mathfrak{n} = \nu n$

Compressed string comparison

Threshold approximate matching



Threshold approx matching (plain pattern, GC text): the algorithm

Algorithm structure similar to local subsequence recognition

$\ll$-chains replaced by $m \times m$ submatrices

Extra ingredients:

- the blow-up technique: reduction of edit distances to LCS scores
- implicit matrix searching, replaces $\ll$-chain interleaving

Monge row minima: "SMAWK" $O(m)$ [Aggarwal+: 1987]

Implicit unit-Monge row minima:

$O(m \log\log m)$ [T: 2012]
$O(m)$ [Gawrychowski: 2012]

Overall time $O(m \log m \cdot \bar{n}) + O(m\bar{n}) = O(m \log m \cdot \bar{n})$

Conclusions and future work

A powerful alternative to dynamic programming

Implicit unit-Monge matrices:

- the seaweed monoid, multiplication in time $O(n \log n)$

Conclusions and future work

A powerful alternative to dynamic programming

Implicit unit-Monge matrices:

- the seaweed monoid, multiplication in time $O(n \log n)$

Semi-local LCS problem:

- isomorphic to seaweed monoid, generalises to rational scores

Conclusions and future work

A powerful alternative to dynamic programming

Implicit unit-Monge matrices:

- the seaweed monoid, multiplication in time $O(n \log n)$

Semi-local LCS problem:

- isomorphic to seaweed monoid, generalises to rational scores

Approximate matching in GC-text in time $O(m \log m \cdot \bar{n})$

Other applications:

- maximum clique in a circle graph in time $O(n \log^2 n)$
- parallel LCS in time $O\left(\frac{mn}{p}\right)$, comm $O\left(\frac{m+n}{p^{1/2}}\right)$ per processor
- identification of evolutionary-conserved regions in DNA

Thank you