Approximate matching in grammar-compressed strings

Alexander Tiskin

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

1. Introduction

2. Semi-local string comparison

3. Matrix distance multiplication

4. Compressed string comparison

5. Conclusions and future work

1 Introduction

2 Semi-local string comparison

3 Matrix distance multiplication

4 Compressed string comparison

5 Conclusions and future work

Introduction

String matching: finding an exact pattern in a string

String comparison: finding similar patterns in two strings

Applications: computational biology, image recognition, ...

Introduction

String matching: finding an exact pattern in a string

String comparison: finding similar patterns in two strings

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

Closely related to approximate string matching (no relation to approximation algorithms!)

Main tool: implicit unit-Monge matrices (a.k.a. seaweed matrices)

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

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

$(i,j) \ll (i',j')$ iff $i < i'$ and $j < j'$ $\quad$ $(i,j) \lessgtr (i',j')$ iff $i < i'$ and $j > j'$

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

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

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

In other words, $D^{\Sigma}(i,j) = \sum D(\hat{\imath},\hat{\jmath})$, where $(\hat{\imath},\hat{\jmath})$ is $\lessgtr$-dominated by (i,j)

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

In other words, $D^{\Sigma}(i,j) = \sum D(\hat{\imath},\hat{\jmath})$, where $(\hat{\imath},\hat{\jmath})$ is $\lessgtr$-dominated by (i,j)

Given matrix E, its density matrix is
$E^{\square}(\hat{\imath},\hat{\jmath}) = E(\hat{\imath}-\frac{1}{2},\hat{\jmath}+\frac{1}{2}) - E(\hat{\imath}-\frac{1}{2},\hat{\jmath}-\frac{1}{2}) - E(\hat{\imath}+\frac{1}{2},\hat{\jmath}+\frac{1}{2}) + E(\hat{\imath}+\frac{1}{2},\hat{\jmath}-\frac{1}{2})$
where D^{Σ}, E over integers; D, $E^{\square}$ over odd half-integers

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

In other words, $D^{\Sigma}(i,j) = \sum D(\hat{\imath},\hat{\jmath})$, where $(\hat{\imath},\hat{\jmath})$ is $\lessgtr$-dominated by (i,j)

Given matrix E, its density matrix is
$E^{\square}(\hat{\imath},\hat{\jmath}) = E(\hat{\imath}-\frac{1}{2},\hat{\jmath}+\frac{1}{2}) - E(\hat{\imath}-\frac{1}{2},\hat{\jmath}-\frac{1}{2}) - E(\hat{\imath}+\frac{1}{2},\hat{\jmath}+\frac{1}{2}) + E(\hat{\imath}+\frac{1}{2},\hat{\jmath}-\frac{1}{2})$

where D^{Σ}, E over integers; D, $E^{\square}$ over odd 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 $D^\Sigma(i,j) = \sum_{\hat\imath > i, \hat\jmath < j} D(\hat\imath, \hat\jmath)$

In other words, $D^\Sigma(i,j) = \sum D(\hat\imath, \hat\jmath)$, where $(\hat\imath, \hat\jmath)$ is $\lessgtr$-dominated by (i,j)

Given matrix E, its density matrix is
$E^\square(\hat\imath, \hat\jmath) = E(\hat\imath - \frac{1}{2}, \hat\jmath + \frac{1}{2}) - E(\hat\imath - \frac{1}{2}, \hat\jmath - \frac{1}{2}) - E(\hat\imath + \frac{1}{2}, \hat\jmath + \frac{1}{2}) + E(\hat\imath + \frac{1}{2}, \hat\jmath - \frac{1}{2})$
where D^Σ, E over integers; D, $E^\square$ over odd 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 $(E^\square)^\Sigma = E$

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

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

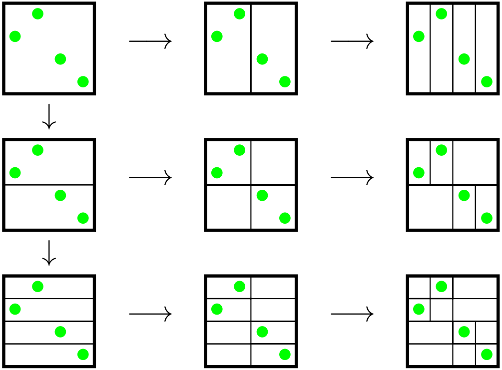
Simple unit-Monge matrix: P^{Σ}, where P is a permutation matrix

Seaweed matrix: P^{Σ}, represented implicitly by P

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

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 means dominance counting: how many nonzeros are dominated by query point? Answered by decomposing query range into $\leq \log^2 n$ disjoint canonical ranges.

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

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 means dominance counting: how many nonzeros are dominated by query 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

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

1 Introduction

2 Semi-local string comparison

3 Matrix distance multiplication

4 Compressed string comparison

5 Conclusions and future work

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 $score(mismatch) = 0$

In biological terms, "loss-free alignment" (unlike "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]

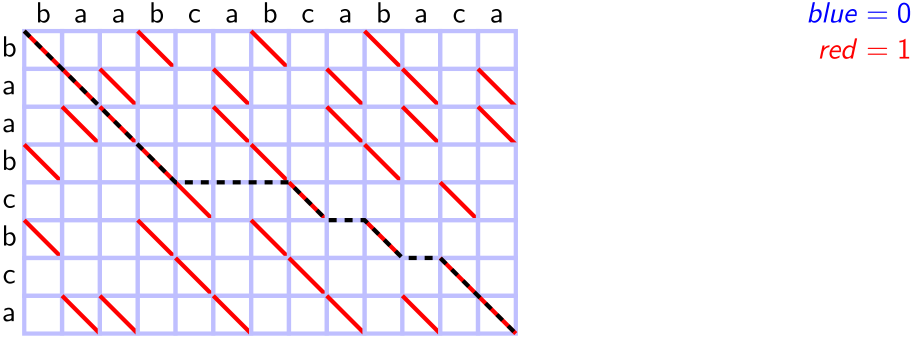
$O\left(\frac{mn(\log\log n)^2}{\log^2 n}\right)$ [Paterson, Dančík: 1994]

[Bille, Farach-Colton: 2008]

Running time varies depending on the RAM model

We assume word-RAM with word size $\log n$

LCS on the alignment graph (directed, acyclic)



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

LCS = highest-score corner-to-corner path

LCS: dynamic programming [WF: 1974]

Sweep alignment graph by cells

Cell update: time $O(1)$

Overall time $O(mn)$

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

Sweep alignment graph by micro-blocks

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 $O(1)$, via table of 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

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

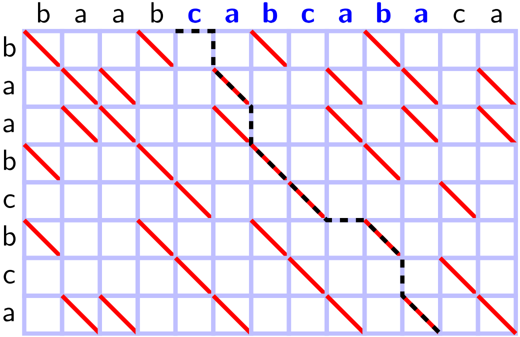
Give the (implicit) matrix of $O(n^2)$ LCS scores:

- 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



blue = 0
red = 1

score("baabcbca", "**cabcaba**") = 5 ("abcba")

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

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"

$b' = b\langle 4 : 11\rangle =$ "**cabcaba**"

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

$H(i, j) = j - i$ 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]

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 $O(\log^2 n)$

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"

$b' = b\langle 4 : 11\rangle =$ "**cabcaba**"

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

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

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"

$b' = b\langle 4:11\rangle =$ "**cabcaba**"

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

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

blue: difference in H is 0

red: difference in H is 1

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"

$b' = b\langle 4 : 11 \rangle =$ "**cabcaba**"

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

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

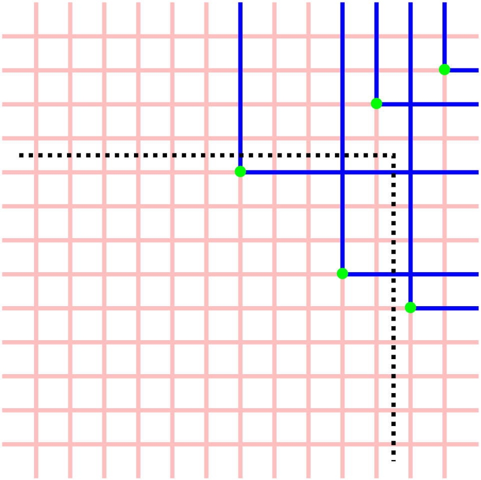
blue: difference in H is 0

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



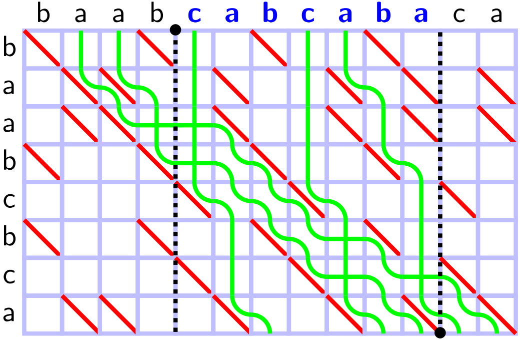
$a =$ "baabcbca"

$b =$ "baab**cabcaba**ca"

$b' = b\langle 4 : 11\rangle =$ "**cabcaba**"

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

The seaweeds in the alignment graph



$a =$ "baabcbca"

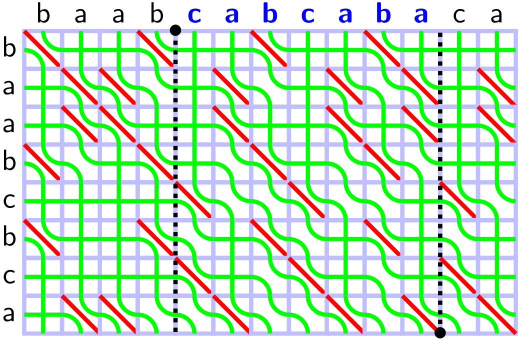
$b =$ "baab**cabcaba**ca"

$b' = b\langle 4:11\rangle =$ "**cabcaba**"

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

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

The seaweeds in the alignment graph



$a =$ "baabcbca"

$b =$ "baab**cabcaba**ca"

$b' = b\langle 4 : 11 \rangle =$ "**cabcaba**"

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

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

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

Gives bijection between top-left and bottom-right borders

1 Introduction

2 Semi-local string comparison

3 Matrix distance multiplication

4 Compressed string comparison

5 Conclusions and future work

Matrix distance multiplication

Seaweed braids

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

Matrix $\odot$-multiplication

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

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

Matrix $\odot$-multiplication

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

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

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

$P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$ written as $P_A \boxdot P_B = P_C$

Matrix distance multiplication

Seaweed braids

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

- simple unit-Monge matrices under $\odot$-multiplication
- permutation matrices under $\boxdot$-multiplication

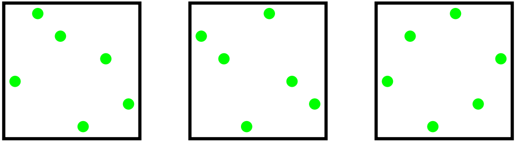
Identity: $1 \boxdot x = x$

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

Zero: $0 \boxdot x = 0$

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

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

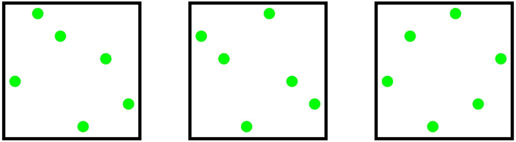


P_A P_B P_C

Matrix distance multiplication

Seaweed braids

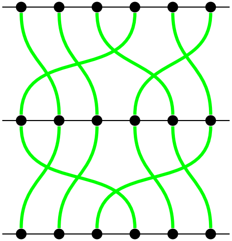
$P_A \boxdot P_B = P_C$ can be seen as $\boxdot$-multiplication of seaweed braids



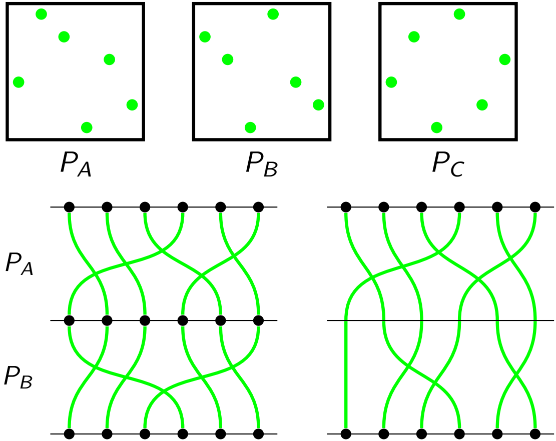
P_A P_B P_C

P_A

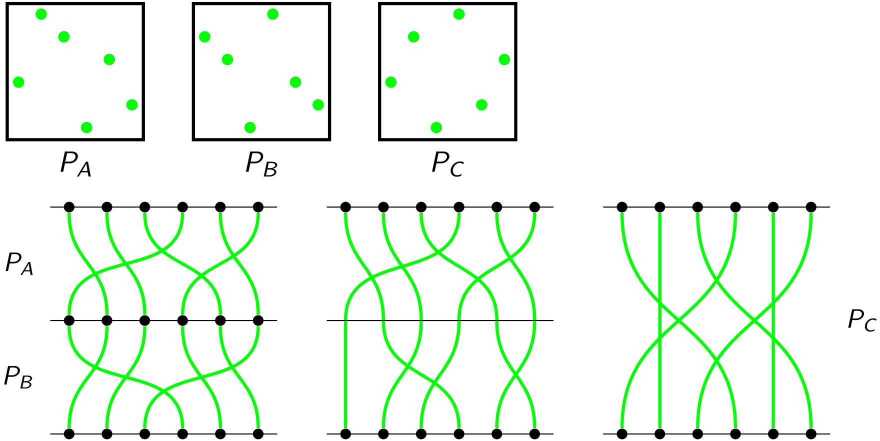
P_B



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



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



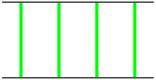
Seaweed braids: similar to standard braids, generated by crossings

Unlike in standard braids, all seaweed crossings are

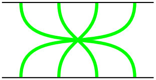
- transversal, i.e. on one level (not underpass/overpass)
- idempotent, i.e. two seaweeds can cross at most once

Seaweed braid $\boxdot$-multiplication: associative, no inverse (a crossing cannot be undone)

Identity: $1 \boxdot x = x$

1


Zero: $0 \boxdot x = 0$

0


Matrix distance multiplication

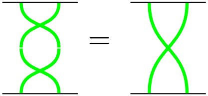
Seaweed braids

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)

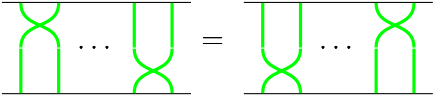
idempotence:

$g_i^2 = g_i$ for all i



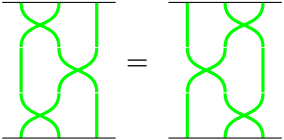
far commutativity:

$g_i g_j = g_j g_i \quad j - i > 1$



braid relations:

$g_i g_j g_i = g_j g_i g_j \quad j - i = 1$



The seaweed monoid $\mathcal{T}_n$

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

Generalisations:

- general 0-Hecke monoids [Fomin, Greene: 1998; Buch+: 2008]
- Coxeter monoids [Tsaranov: 1990; Richardson, Springer: 1990]

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

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, $aba = 0$

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

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, $aba = 0$

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

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

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

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, $aba = 0$

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

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

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

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, $P_A \boxdot P_B = P_C$)

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

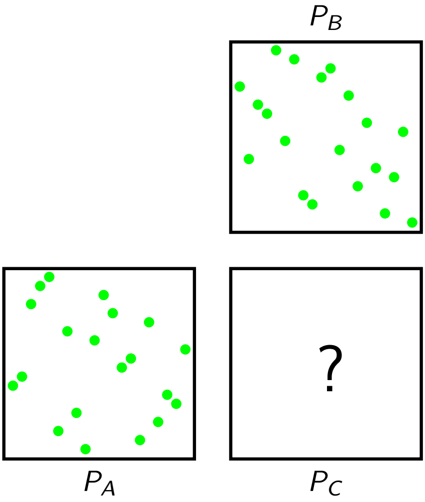
Given permutation matrices P_A, P_B, compute P_C, such that $P_A^\Sigma \odot P_B^\Sigma = P_C^\Sigma$ (equivalently, $P_A \boxdot P_B = P_C$)

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

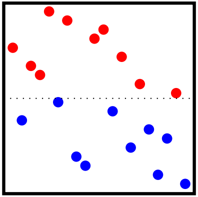
Implicit unit-Monge $\odot$-multiplication

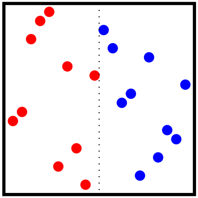


Matrix distance multiplication

Implicit unit-Monge ⊙-multiplication

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

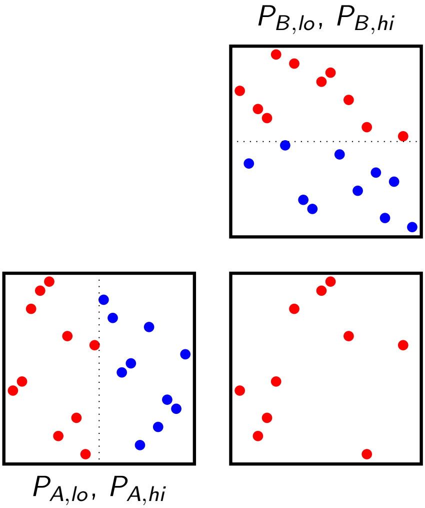




$P_{A,lo}$, $P_{A,hi}$

Matrix distance multiplication

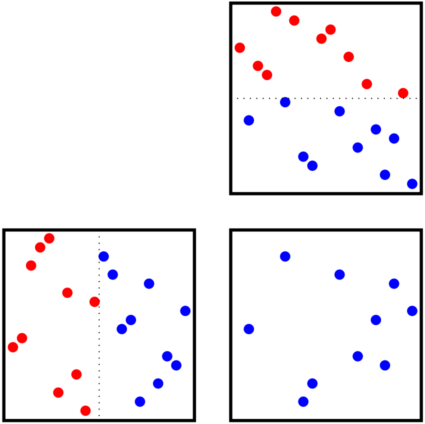
Implicit unit-Monge ⊙-multiplication



Matrix distance multiplication

Implicit unit-Monge ⊙-multiplication

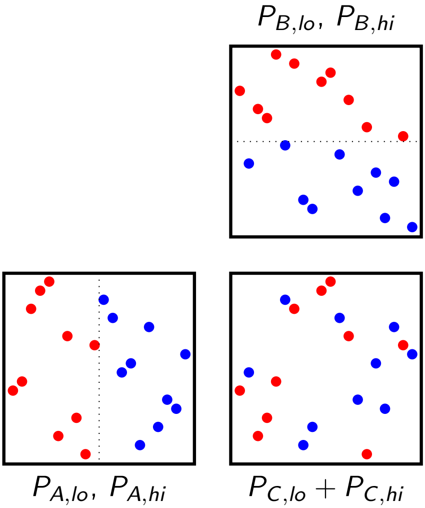
$P_{B,lo}$, $P_{B,hi}$



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

Matrix distance multiplication

Implicit unit-Monge $\odot$-multiplication



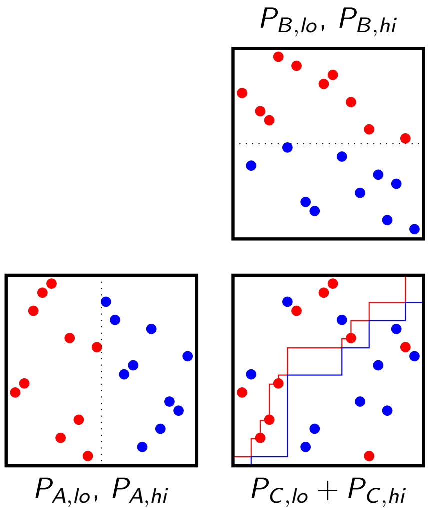
Matrix distance multiplication

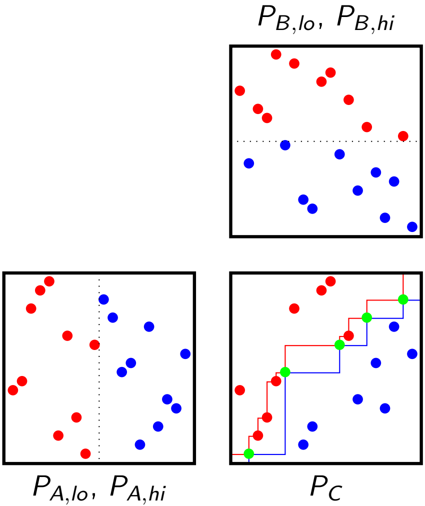
Implicit unit-Monge $\odot$-multiplication

VS

Matrix distance multiplication

Implicit unit-Monge $\odot$-multiplication




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

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

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

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 $O(n \log n)$

1 Introduction

2 Semi-local string comparison

3 Matrix distance multiplication

4 Compressed string comparison

5 Conclusions and future work

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

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
- $t_k = t_i t_j$, where $i, j < k$

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

Example: Fibonacci string "abaababaabaab"

$t_1 = $ 'b' $\quad t_2 = $ 'a'

$t_3 = t_2 t_1 \quad t_4 = t_3 t_2 \quad t_5 = t_4 t_3 \quad t_6 = t_5 t_4 \quad t_7 = t_6 t_5$

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

LCS: running time

t	p			
plain	plain	$O(mn)$		[Wagner, Fischer: 1974]
		$O\left(\frac{mn}{\log^2 m}\right)$		[Masek, Paterson: 1980]
				[Crochemore+: 2003]
GC	plain	$O(m^3\bar{n} + \ldots)$	general CFG	[Myers: 1995]
		$O(m^{1.5}\bar{n})$	3-way semi	[T: 2008]
		$O(m \log m \cdot \bar{n})$	3-way semi	[T: NEW]
GC	GC	NP-hard		[Lifshits: 2005]
		$O(r^{1.2}\bar{r}^{1.4})$		[Hermelin+: 2009]
		$O(r \log r \cdot \bar{r})$		[T: NEW]

$r = m + n \qquad \bar{r} = \bar{m} + \bar{n}$

Three-way semi-local LCS (GC text, plain pattern): the algorithm

For every k, compute by recursion the three-way seaweed matrix for p vs t_k, using seaweed 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 text t contain pattern p as a subsequence?

Global subsequence recognition: running time

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

The local subsequence recognition problem

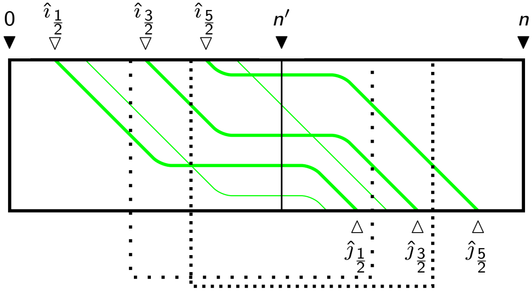
Find all minimally matching substrings of t with respect to p

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

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

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



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

$\lessgtr$-maximal seaweeds: $\ll$-chain $(\hat{\imath}_{\frac{1}{2}}, \hat{\jmath}_{\frac{1}{2}}) \ll (\hat{\imath}_{\frac{3}{2}}, \hat{\jmath}_{\frac{3}{2}}) \ll \cdots \ll (\hat{\imath}_{s-\frac{1}{2}}, \hat{\jmath}_{s-\frac{1}{2}})$

$b\langle i : j\rangle$ minimally matching iff (i, j) is in the interleaved $\ll$-chain
$(\lfloor \hat{\imath}_{\frac{3}{2}} \rfloor, \lceil \hat{\jmath}_{\frac{1}{2}} \rceil) \ll (\lfloor \hat{\imath}_{\frac{5}{2}} \rfloor, \lceil \hat{\jmath}_{\frac{3}{2}} \rceil) \ll \cdots \ll (\lfloor \hat{\imath}_{s-\frac{1}{2}} \rfloor, \lceil \hat{\jmath}_{s-\frac{3}{2}} \rceil)$

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

For every k, compute by recursion the three-way seaweed matrix for p vs t_k, using seaweed matrix $\boxdot$-multiplication: time $O(m \log m \cdot \bar{n})$

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

For every k, compute by recursion the three-way seaweed matrix for p vs t_k, using seaweed matrix $\boxdot$-multiplication: time $O(m \log m \cdot \bar{n})$

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

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

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

For every k, compute by recursion the three-way seaweed matrix for p vs t_k, using seaweed matrix $\boxdot$-multiplication: time $O(m \log m \cdot \bar{n})$

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

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

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

The interleaved $\ll$-chain defines minimally matching substrings in t overlapping both t' and 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*)

t	p		
plain	plain	$O(mn)$	[Sellers: 1980]
		$O(mk)$	[Landau, Vishkin: 1989]
		$O(m + n + \frac{nk^4}{m})$	[Cole, Hariharan: 2002]
GC	plain	$O(m\bar{n}k^2)$	[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)

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

Algorithm structure similar to local subsequence recognition by seaweed matrix $\boxdot$-multiplication and seaweed $\ll$-chains

Extra ingredients:

- the blow-up technique: reduction of edit distances to LCS scores
- the "implicit SMAWK" technique: row minima in an implicit Monge matrix by an extension of the classical "SMAWK" algorithm; replaces $\ll$-chain interleaving

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

1 Introduction

2 Semi-local string comparison

3 Matrix distance multiplication

4 Compressed string comparison

5 Conclusions and future work

Conclusions and future work

A powerful alternative to dynamic programming

Implicit unit-Monge matrices:

- the seaweed monoid
- distance multiplication in time $O(n \log n)$
- next: lower bound?

Conclusions and future work

A powerful alternative to dynamic programming

Implicit unit-Monge matrices:

- the seaweed monoid
- distance multiplication in time $O(n \log n)$
- next: lower bound?

Semi-local LCS problem:

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

Conclusions and future work

A powerful alternative to dynamic programming

Implicit unit-Monge matrices:

- the seaweed monoid
- distance multiplication in time $O(n \log n)$
- next: lower bound?

Semi-local LCS problem:

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