Approximate matching in grammar-compressed strings

Alexander Tiskin

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

2 Matrix distance multiplication

3 Semi-local string comparison

4 Compressed string comparison

5 Conclusions and future work
Introduction

Matrix distance multiplication

Semi-local string comparison

Compressed string comparison

Conclusions and future work
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
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 strings (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)
- \((i, j) \geq (i', j')\) 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 $\geq$-dominance sums:

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

$$D^\Sigma(i, j) = \sum_{\hat{i} > i, \hat{j} < j} D(i, j)$$

Given matrix $E$, its **density matrix** is made up of **quadrangle differences**:

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

where $D^\Sigma, E$ over integers; $D, E^{\square}$ over half-integers
Introduction
Terminology and notation

Given matrix $D$, its distribution matrix is made up of $\geq$-dominance sums:
$$D^\Sigma(i, j) = \sum_{i', j' < j} D(i', j)$$

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

where $D^\Sigma, E$ over integers; $D, E^\square$ over half-integers

$$
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}^\Sigma \\
\begin{bmatrix}
0 & 1 & 2 & 3 \\
0 & 1 & 1 & 2 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}^\square = \\
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}
$$
Given matrix $D$, its distribution matrix is made up of $\geq$-dominance sums:

$$D^\Sigma(i, j) = \sum_{\hat{i}>i, \hat{j}<j} D(i, j)$$

Given matrix $E$, its density matrix is made up of quadrangle differences:

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

where $D^\Sigma$, $E$ over integers; $D$, $E^\Box$ over half-integers

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

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

$$(D^\Sigma)^\Box = D$$ for all $D$

Matrix $E$ is simple, if $(E^\Box)^\Sigma = E$; 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
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 \textit{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 $E^{\Box}$ is a permutation matrix

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

**Simple unit-Monge matrix**: $P^\Sigma$, where $P$ is a permutation matrix

$P^\Sigma$ sometimes called the **rank function** of $P$

**Seaweed matrix**: $P$ used as an implicit representation of $P^\Sigma$

\[
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}^\Sigma = 
\begin{bmatrix}
0 & 1 & 2 & 3 \\
0 & 1 & 1 & 2 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]
Efficient $P^\Sigma$ queries: range tree on nonzeros of $P$  

- binary search tree by $i$-coordinate
- under every node, binary search tree by $j$-coordinate
Efficient $P^\Sigma$ 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 $P^\Sigma$ query is equivalent to $\geq$-dominance counting: how many nonzeros are $\geq$-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 $P^\Sigma$ 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 $P^\Sigma$ query is equivalent to \textit{\geq-dominance counting}: how many nonzeros are \textit{\geq}-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]
1 Introduction

2 Matrix distance multiplication

3 Semi-local string comparison

4 Compressed string comparison

5 Conclusions and future work
Distance algebra (a.k.a. \((\min, +)\) or tropical algebra):

- addition \(\oplus\) given by \(\min\), multiplication \(\odot\) given by \(+\)

Matrix \(\odot\)-multiplication

\[ A \odot B = C \quad C(i, k) = \bigoplus_j (A(i, j) \odot B(j, k)) = \min_j (A(i, j) + B(j, k)) \]

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

The seaweed monoid $T_n$:

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

Also known as the 0-Hecke monoid of the symmetric group $H_0(S_n)$
$P_A \square P_B = P_C$ can be seen as **combing** of seaweed braids.

$P_A \square P_B = P_C$
Matrix distance multiplication

Seaweed braids

\[ P_A \boxtimes P_B = P_C \] can be seen as **combing of seaweed braids**

\[ P_A \boxtimes P_B = P_C \]
Matrix distance multiplication

Seaweed braids

\( P_A \square P_B = P_C \) 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.
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:
\[ g_i^2 = g_i \quad \text{for all } i \]

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: $1 \boxtimes x = x$

$$1 = \begin{bmatrix} * & * & * & * \end{bmatrix} = \begin{array}{l} \text{Diagram 1} \end{array}$$

Zero: $0 \boxtimes x = 0$

$$0 = \begin{bmatrix} * & * & * & * \end{bmatrix} = \begin{array}{l} \text{Diagram 2} \end{array}$$
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 \( S_n \): \( g_i^2 = 1 \); 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]
- \( J \)-trivial monoids [Denton+: 2011]
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)

\[ T_3: 1, a = g_1, b = g_2; \ ab, \ ba; \ aba = 0 \]

\[
\begin{align*}
aa &\to a \\
bb &\to b \\
abab &\to 0 \\
aba &\to 0
\end{align*}
\]
Computation in the seaweed monoid: a **confluent rewriting system** can be obtained by software (**Semigroupe**, GAP)

$T_3$: $1, a = g_1, b = g_2; ab, ba; aba = 0$

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

$T_4$: $1, a = g_1, b = g_2, c = g_3; ab, ac, ba, bc, cb, aba, abc, acb, bac, bcb, cba, abac, abcb, acba, bacb, bcba, abacba, abcba, bacba; abacba = 0$

$aa \rightarrow a$ 
$cb \rightarrow ac$ 
$bab \rightarrow aba$ 
$cbac \rightarrow bcba$

$bb \rightarrow b$ 
$cc \rightarrow c$ 
$cabc \rightarrow bcb$ 
$abacba \rightarrow 0$
Computation in the seaweed monoid: a confluent rewriting system can be obtained by software (Semigroups, GAP)

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

$aa \rightarrow a$  \hspace{1cm}  $bb \rightarrow b$  \hspace{1cm}  $bab \rightarrow 0$  \hspace{1cm}  $aba \rightarrow 0$

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

$aa \rightarrow a$  \hspace{1cm}  $ca \rightarrow ac$  \hspace{1cm}  $bab \rightarrow aba$  \hspace{1cm}  $cbac \rightarrow bcba$

$bb \rightarrow b$  \hspace{1cm}  $cc \rightarrow c$  \hspace{1cm}  $cbc \rightarrow bcb$  \hspace{1cm}  $abacba \rightarrow 0$

Easy to use, but not an efficient algorithm
The implicit unit-Monge matrix $\otimes$-multiplication problem

Given permutation matrices $P_A, P_B$, compute $P_C$, such that

$P_A^\Sigma \otimes P_B^\Sigma = P_C^\Sigma$ (equivalently, $P_A \boxtimes P_B = P_C$)
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 $P_A \odot P_B = P_C$ (equivalently, $P_A \boxdot P_B = P_C$)

Matrix $\odot$-multiplication: running time

<table>
<thead>
<tr>
<th>type</th>
<th>time</th>
<th>time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>$O(n^3)$</td>
<td>standard</td>
</tr>
<tr>
<td></td>
<td>$O\left(\frac{n^3(\log \log n)^3}{\log^2 n}\right)$</td>
<td>[Chan: 2007]</td>
</tr>
<tr>
<td>Monge</td>
<td>$O(n^2)$</td>
<td>via [Aggarwal+ : 1987]</td>
</tr>
<tr>
<td>implicit unit-Monge</td>
<td>$O(n^{1.5})$</td>
<td>[T: 2006]</td>
</tr>
<tr>
<td></td>
<td>$O(n \log n)$</td>
<td>[T: 2010]</td>
</tr>
</tbody>
</table>
Matrix distance multiplication
Seaweed matrix multiplication

$P_B$

$P_A$

$P_C$
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

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

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

\[ P_{B,\text{lo}}, P_{B,\text{hi}} \]

\[ P_{A,\text{lo}}, P_{A,\text{hi}} \]

\[ P_{C} \]
Matrix distance multiplication

Seaweed matrix multiplication

$P_B$

$P_A$

$P_C$
Matrix distance multiplication

Seaweed matrix multiplication

\[ P_A \]

\[ P_B \]

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

$$P_{C}^{\Sigma}(i, k) = \min_{j}(P_{A}^{\Sigma}(i, j) + P_{B}^{\Sigma}(j, k))$$

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

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

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 $O(n \log n)$
Matrix distance multiplication

Bruhat order

Bruhat order

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

Permutation matrices: $P_A \preceq P_B$, if $P_B \leadsto P_A$ by successive $2 \times 2$ submatrix sorting: $(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}) \to (\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix})$
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 $2 \times 2$ submatrix sorting: $
\begin{pmatrix}
0 & 1 \\
1 & 0
\end{pmatrix} \rightarrow
\begin{pmatrix}
1 & 0 \\
0 & 1
\end{pmatrix}$

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

Describes pivoting order in Gaussian elimination (LUP/Bruhat decomposition).
## Bruhat comparability: running time

<table>
<thead>
<tr>
<th>Time Complexity</th>
<th>Author(s) and Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O(n^2))</td>
<td>[Ehresmann: 1934; Proctor: 1982; Grigoriev: 1982]</td>
</tr>
<tr>
<td>(O(n \log n))</td>
<td>[T: 2013]</td>
</tr>
<tr>
<td>(O\left(\frac{n \log n}{\log \log n}\right))</td>
<td>[Gawrychowski: NEW]</td>
</tr>
</tbody>
</table>
Ehresmann's criterion (dot criterion, related to tableau criterion)

\[ P_A \preceq P_B \text{ iff } P_A^\Sigma \leq P_B^\Sigma \text{ elementwise} \]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}^\Sigma =
\begin{bmatrix}
0 & 1 & 2 & 3 \\
0 & 0 & 1 & 2 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix}
\land
\begin{bmatrix}
0 & 1 & 2 & 3 \\
0 & 1 & 2 & 2 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}^\Sigma
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}^\Sigma =
\begin{bmatrix}
0 & 1 & 2 & 3 \\
0 & 0 & 1 & 2 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix}
\land
\begin{bmatrix}
0 & 1 & 2 & 3 \\
0 & 1 & 1 & 2 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix} =
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}^\Sigma
\]

Time \( O(n^2) \)
Seaweed criterion

$P_A \preceq P_B$ iff $P_A^R \boxtimes P_B = Id^R$, where $P^R = \text{clockwise rotation of matrix } P$

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, $P_A^R \boxtimes P_B = Id^R$

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

Bruhat order

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix} \preccurlyeq
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}^R
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix} =
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix} =
\begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0
\end{bmatrix} = Id^R
\]

\[P_A\]
Matrix distance multiplication
Bruhat order

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
\end{bmatrix}
\preceq
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
\end{bmatrix}
\overset{R}{\square}
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{bmatrix}
= \begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0 \\
\end{bmatrix}
\overset{R}{\square}
\begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
\end{bmatrix}
= Id^R
\]

\[
P_A
\]
\[
P_B
\]
Matrix distance multiplication

Bruhat order

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix} \preceq \begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}^R \begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix} = \begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix} \begin{bmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{bmatrix} = \begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0
\end{bmatrix} = \text{Id}^R
\]
Matrix distance multiplication

Bruhat order

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
\end{pmatrix} \preceq \begin{pmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
\end{pmatrix}^R \begin{pmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{pmatrix} = \begin{pmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0 \\
\end{pmatrix} \begin{pmatrix}
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{pmatrix} = \begin{pmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
\end{pmatrix} = \text{Id}^R
\]
Matrix distance multiplication
Bruhat order

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix} \not\preceq \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix} \otimes \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix} \otimes \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix} \neq \text{Id}^R
\]
Matrix distance multiplication

Bruhat order

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix} \not\preceq \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}^R \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix} \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix} \neq \text{Id}^R
\]

\[
\begin{array}{c}
P_A \\
P_B
\end{array}
\]
Matrix distance multiplication

Bruhat order

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix} \not\preceq
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}^R
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
= 
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
= 
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix} \neq Id^R
\]
Matrix distance multiplication

Bruhat order

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}
\not\leq
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}^R
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix} =
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix} =
\begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix} \neq \text{Id}^R
\]
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 $O\left(\frac{n \log n}{\log \log n}\right)$?
1. Introduction

2. Matrix distance multiplication

3. Semi-local string comparison

4. Compressed string comparison

5. Conclusions and future work
Semi-local string comparison
Semi-local LCS and edit distance

Consider strings (= sequences) over an alphabet of size $\sigma$

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 $\sigma$

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 $score(\text{match}) = 1$ and $score(\text{mismatch}) = 0$

In biological terms, “loss-free alignment” (unlike efficient but “lossy” BLAST)
The LCS problem

Give the LCS score for $a$ vs $b$
Semi-local string comparison
Semi-local LCS and edit distance

The LCS problem
Give the LCS score for \(a\) vs \(b\)

LCS: running time

\[
O(mn) \quad \sigma = O(1) \\
O\left(\frac{mn}{\log^2 n}\right) \\
O\left(\frac{mn(\log \log n)^2}{\log^2 n}\right)
\]

[Wagner, Fischer: 1974]
[Masek, Paterson: 1980]
[Crochemore+: 2003]
[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)
Semi-local string comparison

Semi-local LCS and edit distance

LCS computation by dynamic programming

\[
lcs(a, \emptyset) = 0 \\
lcs(\emptyset, b) = 0 \\
lcs(a\alpha, b\beta) = \begin{cases} 
\max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\
lcs(a, b) + 1 & \text{if } \alpha = \beta 
\end{cases}
\]
Semi-local string comparison
Semi-local LCS and edit distance

LCS computation by dynamic programming

\[ lcs(a, \emptyset) = 0 \]
\[ lcs(\emptyset, b) = 0 \]

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

\[ lcs("DEFINE", "DESIGN") = 4 \]
Semi-local string comparison
Semi-local LCS and edit distance

LCS computation by dynamic programming

\[ lcs(a, \emptyset) = 0 \]
\[ lcs(\emptyset, b) = 0 \]
\[ lcs(a\alpha, b\beta) = \begin{cases} 
\max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\
\lcs(a, b) + 1 & \text{if } \alpha = \beta 
\end{cases} \]

<table>
<thead>
<tr>
<th></th>
<th>*</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>I</th>
<th>N</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \lcs("DEFINE", "DESIGN") = 4 \]
Semi-local string comparison
Semi-local LCS and edit distance

LCS computation by **dynamic programming**

\[
lcs(a, \emptyset) = 0
\]

\[
lcs(\emptyset, b) = 0
\]

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

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>I</th>
<th>N</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LCS computation by **dynamic programming**

\[
lcs(a, \emptyset) = 0
\]
\[
lcs(\emptyset, b) = 0
\]
\[
lcs(a\alpha, b\beta) = \begin{cases} 
\max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\
lcs(a, b) + 1 & \text{if } \alpha = \beta 
\end{cases}
\]

<table>
<thead>
<tr>
<th></th>
<th>*</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>I</th>
<th>N</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Semi-local string comparison

Semi-local LCS and edit distance

LCS computation by **dynamic programming**

\[
lcs(a, \emptyset) = 0 \quad \text{and} \quad lcs(\emptyset, b) = 0
\]

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

\[
\begin{array}{cccccccc}
* & D & E & F & I & N & E \\
* & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
D & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
E & 0 & 1 & 2 & 2 & 2 & 2 & 2 \\
S & 0 & 1 & 2 & 2 & 2 & 2 & 2 \\
I & 0 & 1 & 2 & 2 & 3 & 3 & 3 \\
G & 0 & & & & & & \\
N & 0 & & & & & & \\
\end{array}
\]
Semi-local string comparison
Semi-local LCS and edit distance

LCS computation by **dynamic programming**

\[ lcs(a, \emptyset) = 0 \]
\[ lcs(\emptyset, b) = 0 \]
\[ lcs(a\alpha, b\beta) = \begin{cases} 
    \max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\
    lcs(a, b) + 1 & \text{if } \alpha = \beta 
\end{cases} \]

<table>
<thead>
<tr>
<th></th>
<th>*</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>I</th>
<th>N</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Semi-local string comparison

Semi-local LCS and edit distance

LCS computation by dynamic programming

\[ lcs(a, \emptyset) = 0 \]
\[ lcs(\emptyset, b) = 0 \]
\[ lcs(a\alpha, b\beta) = \begin{cases} 
\max(lcs(a\alpha, b), lcs(a, b\beta)) & \text{if } \alpha \neq \beta \\
lcs(a, b) + 1 & \text{if } \alpha = \beta 
\end{cases} \]

<table>
<thead>
<tr>
<th></th>
<th>*</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>I</th>
<th>N</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ lcs(\text{"DEFINE"}, \text{"DESIGN"}) = 4 \]
Semi-local string comparison
Semi-local LCS and edit distance

LCS computation by **dynamic programming**

\[
lcs(a, \emptyset) = 0 \quad \text{and} \quad lcs(\emptyset, b) = 0
\]

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

<table>
<thead>
<tr>
<th></th>
<th>*</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>I</th>
<th>N</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

\[
lcs(\text{"DEFINE"}, \text{"DESIGN"}) = 4
\]

\(LCS(a, b)\) can be traced back through the dynamic programming table at no extra asymptotic time cost.
Semi-local string comparison
Semi-local LCS and edit distance

LCS on the alignment graph (directed, acyclic)

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

LCS = highest-score path from top-left to bottom-right
Semi-local string comparison
Semi-local LCS and edit distance

LCS: dynamic programming [WF: 1974]

- Sweep cells in any $\preceq$-compatible order
- Cell update: time $O(1)$
- Overall time $O(mn)$
Semi-local string comparison
Semi-local LCS and edit distance


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 $b$
Semi-local string comparison
Semi-local LCS and edit distance

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 string comparison
Semi-local LCS and edit distance

Semi-local LCS on the alignment graph

\[ \text{score}(\text{"BAABCBBCA"}, \text{"CABCABA"}) = \text{len}(\text{"ABCBA"}) = 5 \]

String-substring LCS: all highest-score top-to-bottom paths
Semi-local LCS: all highest-score boundary-to-boundary paths
Semi-local string comparison
Score matrices and seaweed matrices

The score matrix $H$

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 6 & 7 & 8 & 8 & 8 & 8 & 8 \\
-1 & 0 & 1 & 2 & 3 & 4 & 5 & 5 & 6 & 7 & 7 & 7 & 7 & 7 \\
-2 & -1 & 0 & 1 & 2 & 3 & 4 & 4 & 5 & 6 & 6 & 6 & 6 & 7 \\
-3 & -2 & -1 & 0 & 1 & 2 & 3 & 3 & 4 & 5 & 5 & 6 & 6 & 7 \\
-4 & -3 & -2 & -1 & 0 & 1 & 2 & 2 & 3 & 4 & 4 & 4 & 5 & \textcircled{5} \\
-5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 & 4 & 5 & 5 & 6 \\
-6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 3 & 4 & 4 & 5 \\
-7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 3 & 3 & 4 \\
-8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 3 & 4 \\
-9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 4 \\
-10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 \\
-11 & -10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 \\
-12 & -11 & -10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 \\
-13 & -12 & -11 & -10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 \\
\end{array}
\]

\[
a = \text{“BAABCBCA”}
\]
\[
b = \text{“BAABCABCA”}
\]
\[
H(i,j) = \text{score}(a, b\langle i : j \rangle)
\]
\[
H(4,11) = 5
\]
\[
H(i,j) = j - i \text{ if } i > j
\]
Semi-local string comparison
Score matrices and seaweed matrices

Semi-local LCS: output representation and running time

<table>
<thead>
<tr>
<th>size</th>
<th>query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n^2)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>$O(m^{1/2} n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>$O(\log^2 n)$</td>
</tr>
</tbody>
</table>

...or any 2D orthogonal range counting data structure

running time

<table>
<thead>
<tr>
<th>$O(mn^2)$</th>
<th>naive</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(mn)$</td>
<td>string-substring [Schmidt: 1998; Alves+: 2005]</td>
</tr>
<tr>
<td>$O(mn)$</td>
<td>[T: 2006]</td>
</tr>
<tr>
<td>$O\left(\frac{mn}{\log^{0.5} n}\right)$</td>
<td>[T: 2006]</td>
</tr>
<tr>
<td>$O\left(\frac{mn(\log \log n)^2}{\log^2 n}\right)$</td>
<td>[T: 2007]</td>
</tr>
</tbody>
</table>
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 $b(i : j)$

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

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

- simple unit-Monge
- therefore, $= P^Σ$, where $P = -H$ 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 $P$

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 8 & 8 & 8 \\
-1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 7 & 7 & 7 \\
-2 & -1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 6 & 6 & 6 \\
-3 & -2 & -1 & 0 & 1 & 2 & 3 & 3 & 4 & 5 & 5 & 6 \\
-4 & -3 & -2 & -1 & 0 & 1 & 2 & 2 & 3 & 4 & 4 & 5 \\
-5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 3 & 4 & 4 \\
-6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 2 & 3 & 3 \\
-7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & 3 \\
-8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 \\
-9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 & 2 \\
-10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 & 1 \\
-11 & -10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 & 0 \\
-12 & -11 & -10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 & -1 \\
-13 & -12 & -11 & -10 & -9 & -8 & -7 & -6 & -5 & -4 & -3 & -2 \\
\end{array}
\]

\[a = "BAABCBCA"
\]
\[b = "BAABCABCABCA"
\]
\[H(i, j) = score(a, b^{i:j})
\]
\[H(4, 11) = 5
\]
\[H(i, j) = j - i \text{ if } i > j
\]
The score matrix $H$ and the seaweed matrix $P$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>8</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>-11</td>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>-12</td>
<td>-11</td>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>-13</td>
<td>-12</td>
<td>-11</td>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

$a = \text{“BAABCBACA”}$

$b = \text{“BAABCABCABACA”}$

$H(i, j) = \text{score}(a, b⟨i : j⟩)$

$H(4, 11) = 5$

$H(i, j) = j - i \text{ if } i > j$

*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 $P$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>8</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>-11</td>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-12</td>
<td>-11</td>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>-13</td>
<td>-12</td>
<td>-11</td>
<td>-10</td>
<td>-9</td>
<td>-8</td>
<td>-7</td>
<td>-6</td>
<td>-5</td>
<td>-4</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
</tr>
</tbody>
</table>

$a = \text{“BAABCBCA”}$

$b = \text{“BAABCABCABACA”}$

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

$H(4, 11) = 5$

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

blue: difference in $H$ is 0

red: difference in $H$ is 1

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

$H(i, j) = j - i - P^\Sigma(i, j)$
Semi-local string comparison
Score matrices and seaweed matrices

The score matrix $H$ and the seaweed matrix $P$

$a = \text{“BAABCBCA”}$

$b = \text{“BAABCABCABACA”}$

$H(4, 11) =
11 - 4 - P_{\Sigma}(i, j) =
11 - 4 - 2 = 5$
Semi-local string comparison
Score matrices and seaweed matrices

The (combed) **seaweed braid** in the alignment graph

\[
a = \text{"BAABCBCA"}
\]
\[
b = \text{"BAABCABCABACA"}
\]
\[
H(4, 11) = 11 - 4 - P^\Sigma(i, j) = 11 - 4 - 2 = 5
\]

\[P(i, j) = 1\text{ corresponds to seaweed top } i \rightsquigarrow \text{ bottom } j\]
The (combed) **seaweed braid** in the alignment graph

\[ a = \text{“BAABCBCA”} \]
\[ b = \text{“BAABBABCABABCBA”} \]
\[ H(4, 11) = \]
\[ 11 - 4 - P_{\Sigma}(i, j) = \]
\[ 11 - 4 - 2 = 5 \]

\[ P(i, j) = 1 \] corresponds to seaweed **top** \( i \mapsto \text{bottom} \) \( j \)

Also define seaweeds **top** \( \mapsto \text{right} \), **left** \( \mapsto \text{right} \), **left** \( \mapsto \text{bottom} \)

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: $w_M = 2$, $w_X = 1$, $w_G = 0$
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 \(-w_X\)
Semi-local string comparison

Weighted alignment

Weighted alignment graph for $a$, $b$

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

$blue = 0$

$red$ (solid) = 2

$red$ (dotted) = 1
Semi-local string comparison
Weighted alignment

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

$B \ A \ A \ B \ C \ A \ B \ C \ A \ B \ A \ C \ A$

$B$
$A$
$A$
$B$
$C$
$B$
$C$
$A$

$blue = 0$
$red = 1 \ or \ 2$

Levenshtein("BAABCBCA", "CABCABA") =
lcs("$B$A$A$B$C$B$C$A$", "$C$A$B$C$A$B$A$") = 11
Semi-local string comparison

Weighted alignment

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

$B \ A \ A \ B \ C \ A \ B \ C \ A \ B \ A \ C \ A$

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

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

2 Matrix distance multiplication

3 Semi-local string comparison

4 Compressed string comparison

5 Conclusions and future work
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 $\alpha$ 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 “ABAABABABAABAAB”

$t_1 = A \quad t_2 = t_1B \quad t_3 = t_2t_1 \quad t_4 = t_3t_2 \quad t_5 = t_4t_3 \quad t_6 = t_5t_4$
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 \((r = m + n, \bar{r} = \bar{m} + \bar{n})\)

| p   | t   | \(O(mn)\) | \(O(\frac{mn}{\log^2 m})\) | \(O(m^3\bar{n} + \ldots)\) | gen. CFG | \(O(m^{1.5}\bar{n})\) | ext subs-s | \(O(m \log m \cdot \bar{n})\) | ext subs-s | \(O(r^{1.2}\bar{r}^{1.4})\) | R weights | \(O(r \log r \cdot \bar{r})\) | \(O(r \log(r/\bar{r}) \cdot \bar{r})\) | \(O(r \log^{1/2}(r/\bar{r}) \cdot \bar{r})\) |
|-----|-----|-----------|----------------------------|----------------------------|---------|----------------|-------------|----------------|-------------|----------------|-------------|----------------|-------------|----------------|-------------|----------------|
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 $\Box$-multiplication: time $O(m \log m \cdot \bar{n})$

Overall time $O(m \log m \cdot \bar{n})$
Compressed string comparison
Subsequence recognition on GC-strings

The global subsequence recognition problem

Does pattern $p$ appear in text $t$ as a subsequence?

Global subsequence recognition: running time

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$t$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>plain</td>
<td>plain</td>
<td>$O(n)$</td>
<td>greedy</td>
</tr>
<tr>
<td>plain</td>
<td>GC</td>
<td>$O(m\tilde{n})$</td>
<td>greedy</td>
</tr>
<tr>
<td>GC</td>
<td>GC</td>
<td>NP-hard</td>
<td>[Lifshits: 2005]</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>$p$</th>
<th>$t$</th>
<th>$O$ expression</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>plain</td>
<td>plain</td>
<td>$O(mn)$</td>
<td>[Mannila+: 1995]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O\left(\frac{mn}{\log m}\right)$</td>
<td>[Das+: 1997]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(c^m + n)$</td>
<td>[Boasson+: 2001]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(m + n\sigma)$</td>
<td>[Troniček: 2001]</td>
</tr>
<tr>
<td>plain</td>
<td>GC</td>
<td>$O(m^2 \log m\tilde{n})$</td>
<td>[Cégielski+: 2006]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(m^{1.5}\tilde{n})$</td>
<td>[T: 2008]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(m \log m \cdot \tilde{n})$</td>
<td>[T: 2010]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(m \cdot \tilde{n})$</td>
<td>[Yamamoto+: 2011]</td>
</tr>
<tr>
<td>GC</td>
<td>GC</td>
<td>NP-hard</td>
<td>[Lifshits: 2005]</td>
</tr>
</tbody>
</table>
Compressed string comparison
Subsequence recognition on GC-strings

\[ b(i : j) \text{ matching iff box } [i : j] \text{ not pierced left-to-right} \]

Determined by \( \leq \)-chain of \( \geq \)-maximal seaweeds

\[ b(i : j) \text{ minimally matching iff } (i, j) \text{ is in the interleaved skeleton } \leq \text{-chain} \]
Compressed string comparison

Subsequence recognition on GC-strings

Diagram showing the relationship between $m$, $n'$, $n$, and $m+n$ with points labeled as $\hat{i}_{0+}$, $\hat{i}_{1+}$, and $\hat{i}_{2+}$.
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 $\Box$-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 $\square$-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 $\bullet$-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 $\gg$-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>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$p$</td>
<td>$t$</td>
<td></td>
</tr>
<tr>
<td><strong>plain</strong></td>
<td><strong>plain</strong></td>
<td>$O(mn)$</td>
<td>[Sellers: 1980]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(mk)$</td>
<td>[Landau, Vishkin: 1989]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(m + n + \frac{nk^4}{m})$</td>
<td>[Cole, Hariharan: 2002]</td>
<td></td>
</tr>
<tr>
<td><strong>plain</strong></td>
<td><strong>GC</strong></td>
<td>$O(m\bar{n}k^2)$</td>
<td>[Kärkkäinen+: 2003]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(m\bar{n}k + \bar{n}\log n)$</td>
<td>[LV: 1989] via [Bille+: 2010]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(m\bar{n} + \bar{n}k^4 + \bar{n}\log n)$</td>
<td>[CH: 2002] via [Bille+: 2010]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$O(m \log m \cdot \bar{n})$</td>
<td>[T: NEW]</td>
<td></td>
</tr>
<tr>
<td><strong>GC</strong></td>
<td><strong>GC</strong></td>
<td>NP-hard</td>
<td>[Lifshits: 2005]</td>
<td></td>
</tr>
</tbody>
</table>

(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 $p$, $t$ of size $m = \nu m$, $n = \nu n$
Compressed string comparison

Threshold approximate matching

\[ \hat{i}_0^+, \hat{i}_1^+, \hat{i}_2^+, \hat{i}_3^+, \hat{i}_4^+ \]

\[ n', \hat{j}_0^+, \hat{j}_1^+, \hat{j}_2^+, \hat{j}_3^+, \hat{j}_4^+ \]
Threshold approx matching (plain pattern, GC text): the algorithm

Algorithm structure similar to local subsequence recognition

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

Extra ingredients:

- the **blow-up** technique: reduction of edit distances to LCS scores
- implicit matrix searching, replaces ≪-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 $O(m \log m \cdot \bar{n}) + O(m \bar{n}) = O(m \log m \cdot \bar{n})$
1 Introduction

2 Matrix distance multiplication

3 Semi-local string comparison

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, 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

Approximate matching in GC-strings
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