UNIVERSITÄT
DES
SAARLANDES

ZBI ZENTRUM FÜR
BIOINFORMATIK

Error Tolerant Pattern Matching I

Algorithms for Sequence Analysis

Sven Rahmann

Summer 2021

Overview

Previous Lecture

- Definition of distance measures, in particular **edit distance**
- Sequence of edit operations: copy (free), substitution, insertion, deletion (1 each)
- **Global** sequence alignment (full strings)
- Edit path in edit graph
- Edit distance = least-cost edit sequence
  = least-cost alignment = least-cost path in alignment graph

Overview

Previous Lecture

- Definition of distance measures, in particular **edit distance**
- Sequence of edit operations: copy (free), substitution, insertion, deletion (1 each)
- **Global** sequence alignment (full strings)
- Edit path in edit graph
- Edit distance = least-cost edit sequence
  = least-cost alignment = least-cost path in alignment graph

Today's Lecture

- Pattern Matching with respect to edit distance
- **Semiglobal** Alignment:
  Compare one full string (pattern) against substrings of a longer string (text)

Error Tolerant Pattern Matching

Problem Definition

- For two strings $P, T \in \Sigma^*$, find **approximate occurrences** of P in T.
- **Formally:** Find **intervals** $[i, j]$ of T such that the **edit distance** between P and $T[i \dots j]$ is below a given threshold k.

Error Tolerant Pattern Matching

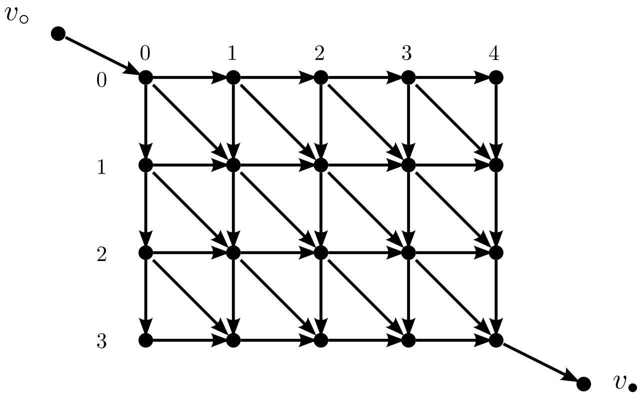
Problem Definition

- For two strings $P, T \in \Sigma^*$, find **approximate occurrences** of P in T.
- **Formally:** Find **intervals** $[i, j]$ of T such that the **edit distance** between P and $T[i \dots j]$ is below a given threshold k.

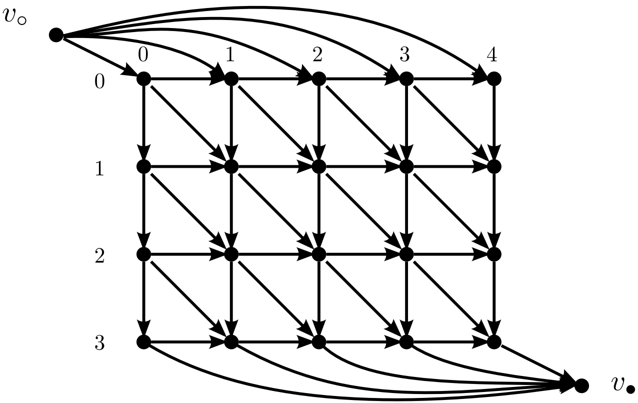
Variants

- **Decision Problem:** Is there an interval . . . ?
- **Counting Problem:** How many intervals . . . ?
- **Enumeration Problem:** List all intervals . . . .
- **Optimization Problem:** Find an interval $[i, j]$ with the smallest edit distance between P and $T[i \dots j]$ among all (no threshold k given).

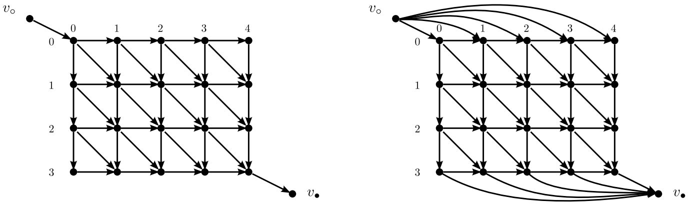
Alignment Graph for Global Alignment



Alignment Graph for Semiglobal Alignment (Pattern Matching)



Generalized Edit Graph Algorithm

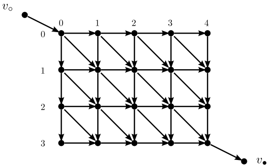


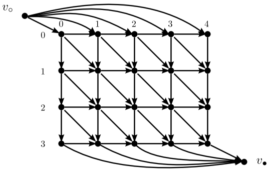
1. **Topologically sort** all nodes
2. **Iterate** over nodes in topological order
   - **Compute value** of that node: **Minimize** over incoming edges: cost at source node + edge cost
   - **Mark** edge(s) that gave rise to optimum
3. Value at final node $v_\bullet$ gives solution of optimization problem
4. **Traceback:** go back along marked edges to construct alignment

Code: Semiglobal Alignment (Enumeration of End Positions)

```python
def pattern_search(P, T, k):
    # compare edit ditance computation
    m, n = len(P), len(T)
    # Column 0
    Dc = list(range(m+1)) # Dc: current column in D
    Dp = [0] * (m+1)  # Dp: previous column in D
    # Iterate over columns j and characters tj=T[j] in T
    for j, tj in zip(count(1), T):
        Dp, Dc = Dc, Dp  # swap to recompute Dc
        Dc[0] = 0  # row 0: D[0,j] = 0, start anywhere
        # iterate over rows i and characters pi=P[i] in P
        for i, pi in zip(count(1), P):
            Dc[i] = min( Dp[i - 1] + (pi != tj),
                         Dp[i] + 1,
                         Dc[i - 1] + 1 )
        if Dc[m] <= k: yield j
```

Traceback: Getting the Actual Alignment

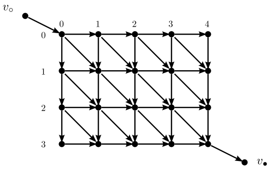


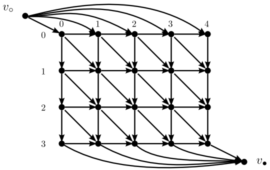


Approach

- When computing the value of a graph node (cell in DP table), **memorize the predecessor(s)** that gave rise to the optimum.
- From bottom right cell, move to top left along these edges.

Traceback: Getting the Actual Alignment

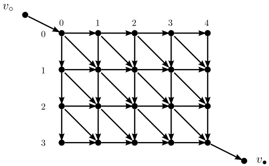


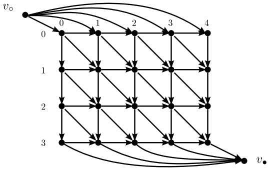


Approach

- When computing the value of a graph node (cell in DP table), **memorize the predecessor(s)** that gave rise to the optimum.
- From bottom right cell, move to top left along these edges.
- Reconstructs the alignment in reverse (but easy to flip when done)

Traceback: Getting the Actual Alignment





Approach

- When computing the value of a graph node (cell in DP table), **memorize the predecessor(s)** that gave rise to the optimum.
- From bottom right cell, move to top left along these edges.
- Reconstructs the alignment in reverse (but easy to flip when done)

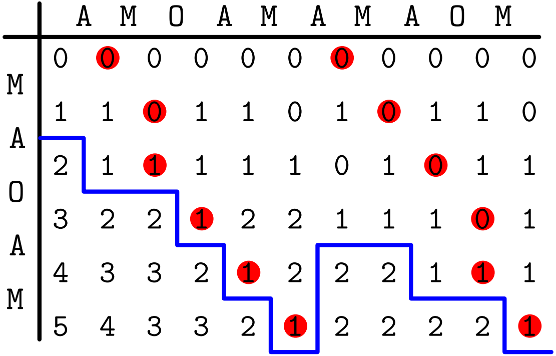
Options for managing the traceback information

- Option 1: **store** "predecessor edges" in **extra table**
- Option 2: **recompute** them for each cell encountered during traceback

Ukkonen's Speed-Up of Semiglobal Alignment

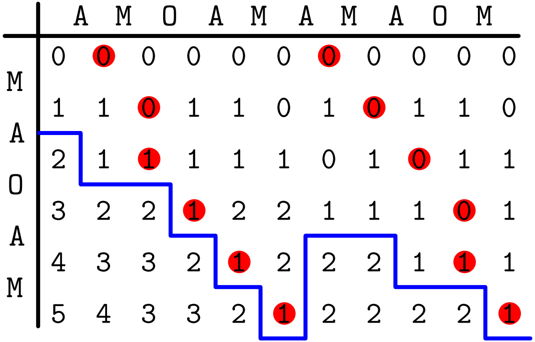
Setting as in shown code

- Search for approximate occurrences with edit distance $\leq k$.
- **Question:** Can we omit computing parts of the graph / DP table?



		A	M	O	A	M	A	M	A	O	M
	0	**0**	0	0	0	0	**0**	0	0	0	0
M	1	1	**0**	1	1	0	1	**0**	1	1	0
A	2	1	**1**	1	1	1	0	1	**0**	1	1
O	3	2	2	**1**	2	2	1	1	1	**0**	1
A	4	3	3	2	**1**	2	2	2	1	**1**	1
M	5	4	3	3	2	**1**	2	2	2	2	**1**

Semiglobal Alignment: Ukkonen's Trick



		A	M	O	A	M	A	M	A	O	M
	0	0	0	0	0	0	0	0	0	0	0
M	1	1	0	1	1	0	1	0	1	1	0
A	2	1	1	1	1	1	0	1	0	1	1
O	3	2	2	1	2	2	1	1	1	0	1
A	4	3	3	2	1	2	2	2	1	1	1
M	5	4	3	3	2	1	2	2	2	2	1

- **Define:** $last_k(j) := \max\{i \,|\, T[i,j] \leq k,\; T[i',j] > k \text{ for all } i' > i\}$
- **Trick:** $last_k(j+1) \leq last_k(j) + 1$
  Why? Difference between neighboring cells is at most 1 (proof follows).
- **Speedup:** average runtime improves from $O(mn)$ to $O(kn)$ (hard proof!)

Edit Distance DP Table: Horizontal, Vertical, and Diagonal Properties

Motivation

Setting

DP table for edit distance computations: global and semi-global alignment

We astated that the **difference of neighboring cells is at most 1**.

Needed for speed-ups:

- Ukkonen's trick (just seen)
- Myers' bit-parallel algorithm (next)

Goal

State precise properties and prove them.

Properties

Let T be a DP table satisfying the edit distance recurrence.

Lemma: Vertical Property

The value difference between any two **vertically adjacent** cells is at most 1: $|T[i,j] - T[i-1,j]| \leq 1$.

Lemma: Horizontal Property

The value difference between any two **horizontally adjacent** cells is at most 1: $|T[i,j] - T[i,j-1]| \leq 1$.

Lemma: Diagonal Property

The value of **diagonally adjacent** cells is non-decreasing,
and the value difference is at most 1, i.e., $0 \leq T[i,j] - T[i-1,j-1] \leq 1$.

Vertical Property

Lemma: Vertical Property

The value difference between any two **vertically adjacent** cells is at most 1: $|T[i,j] - T[i-1,j]| \leq 1$.

Proof idea

Show separately:

- $T[i,j] - T[i-1,j] \leq 1 \Leftrightarrow T[i,j] \leq T[i-1,j] + 1$
- $T[i-1,j] - T[i,j] \leq 1 \Leftrightarrow T[i-1,j] - 1 \leq T[i,j]$

Vertical Property

Lemma: Vertical Property

The value difference between any two **vertically adjacent** cells is at most 1: $|T[i,j] - T[i-1,j]| \leq 1$.

Proof idea

Show separately:

- $T[i,j] - T[i-1,j] \leq 1 \Leftrightarrow T[i,j] \leq T[i-1,j] + 1$
  - Follows immediately from recurrence
- $T[i-1,j] - T[i,j] \leq 1 \Leftrightarrow T[i-1,j] - 1 \leq T[i,j]$
  - Left to be shown.

Vertical Property

Lemma: Vertical Property

The value difference between any two **vertically adjacent** cells is at most 1: $|T[i,j] - T[i-1,j]| \leq 1$.

Proof idea

Show separately:

- $T[i,j] - T[i-1,j] \leq 1 \Leftrightarrow T[i,j] \leq T[i-1,j] + 1$
  - Follows immediately from recurrence
- $T[i-1,j] - T[i,j] \leq 1 \Leftrightarrow T[i-1,j] - 1 \leq T[i,j]$
  - Left to be shown.

d	v	$x \leq v+1$ ✓
h	x	$v-1 \leq x$ (to be shown)

Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

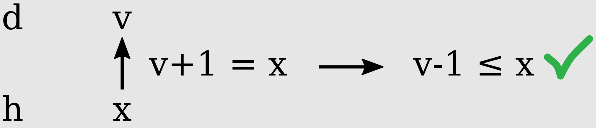
Case 1: Minimum from vertical neighbor

$$\begin{tikzcd} d & v & & \\ & & v+1 = x \arrow[r] & v-1 \leq x \checkmark \\ h & x \arrow[uu] & & \end{tikzcd}$$

Vertical Property (cont'd)

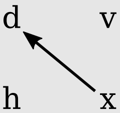
To be shown: $v - 1 \leq x$

Case 1: Minimum from vertical neighbor



$v+1 = x \longrightarrow v-1 \leq x$ ✓

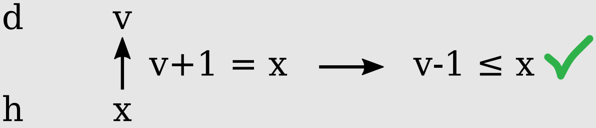
Case 2: Minimum from diagonal neighbor



Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Case 1: Minimum from vertical neighbor

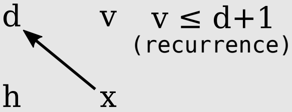


d v

h x

$v+1 = x \longrightarrow v-1 \leq x$ ✓

Case 2: Minimum from diagonal neighbor



d v

h x

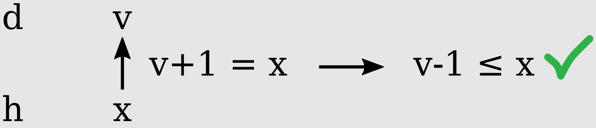
$v \leq d+1$

(recurrence)

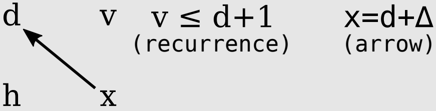
Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Case 1: Minimum from vertical neighbor



Case 2: Minimum from diagonal neighbor



Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd}
d & v \\
h & x \arrow[u]
\end{tikzcd}$$

$v+1 = x \longrightarrow v-1 \leq x$ ✓

Case 2: Minimum from diagonal neighbor

$$\begin{tikzcd}
d & v \\
h & x \arrow[ul]
\end{tikzcd}$$

$v \leq d+1$ (recurrence)

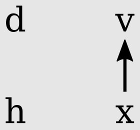
$x=d+\Delta$ (arrow)

$v \leq x-\Delta+1$

Vertical Property (cont'd)

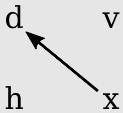
To be shown: $v - 1 \leq x$

Case 1: Minimum from vertical neighbor



$v+1 = x \longrightarrow v-1 \leq x$ ✓

Case 2: Minimum from diagonal neighbor



$v \leq d+1$ (recurrence)

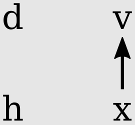
$x=d+\Delta$ (arrow)

$v \leq x-\Delta+1 \leq x+1$

Vertical Property (cont'd)

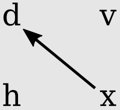
To be shown: $v - 1 \leq x$

Case 1: Minimum from vertical neighbor



$v+1 = x \longrightarrow v-1 \leq x$ ✓

Case 2: Minimum from diagonal neighbor



$v \leq d+1$ (recurrence) $x = d+\Delta$ (arrow)

$v \leq x-\Delta+1 \leq x+1 \longrightarrow v-1 \leq x$ ✓

Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Case 3: Minimum from horizontal neighbor

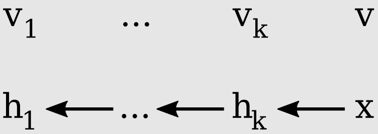
$$\begin{tikzcd}
v_1 & \dots & v_k & v \\
h_1 & \dots \arrow[l] & h_k \arrow[l] & x \arrow[l]
\end{tikzcd}$$

Proof by induction:

Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Case 3: Minimum from horizontal neighbor



v_1 … v_k v

$h_1 \leftarrow \ldots \leftarrow h_k \leftarrow x$

Proof by induction:

- **Induction base ($v_1 - 1 \leq h_1$):** (follows from Case I or II)

Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Case 3: Minimum from horizontal neighbor

$$\begin{tikzcd}
v_1 & \ldots & v_k & v \\
h_1 & \ldots \arrow[l] & h_k \arrow[l] & x \arrow[l]
\end{tikzcd}$$

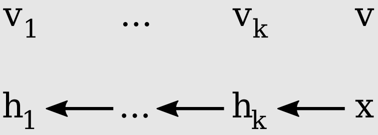
Proof by induction:

- **Induction base ($v_1 - 1 \leq h_1$):** (follows from Case I or II)
- **Induction step ($v_{j-1} - 1 \leq h_{j-1} \Rightarrow v_j - 1 \leq h_j$):** we have:

Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Case 3: Minimum from horizontal neighbor



v_1 … v_k v

$h_1 \longleftarrow \ldots \longleftarrow h_k \longleftarrow x$

Proof by induction:

- **Induction base ($v_1 - 1 \leq h_1$):** (follows from Case I or II)
- **Induction step ($v_{j-1} - 1 \leq h_{j-1} \Rightarrow v_j - 1 \leq h_j$):** we have:
  - $h_{j-1} + 1 = h_j$ (by definition, minimum is horizontal)

Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Case 3: Minimum from horizontal neighbor

$$\begin{tikzcd}
v_1 & \cdots & v_k & v \\
h_1 & \cdots \arrow[l] & h_k \arrow[l] & x \arrow[l]
\end{tikzcd}$$

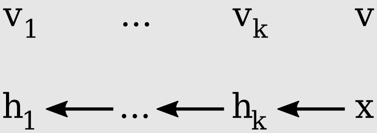
Proof by induction:

- **Induction base ($v_1 - 1 \leq h_1$):** (follows from Case I or II)
- **Induction step ($v_{j-1} - 1 \leq h_{j-1} \Rightarrow v_j - 1 \leq h_j$):** we have:
  - $h_{j-1} + 1 = h_j$ (by definition, minimum is horizontal)
  - $v_{j-1} \leq h_{j-1} + 1$ (by induction assumption)

Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Case 3: Minimum from horizontal neighbor



v_1 … v_k v

h_1 ← … ← h_k ← x

Proof by induction:

- **Induction base ($v_1 - 1 \leq h_1$):** (follows from Case I or II)
- **Induction step ($v_{j-1} - 1 \leq h_{j-1} \Rightarrow v_j - 1 \leq h_j$):** we have:
  - $h_{j-1} + 1 = h_j$ (by definition, minimum is horizontal)
  - $v_{j-1} \leq h_{j-1} + 1$ (by induction assumption)
  - $v_j - 1 \leq v_{j-1}$ (by recurrence relation)

Vertical Property (cont'd)

To be shown: $v - 1 \leq x$

Case 3: Minimum from horizontal neighbor

$$\begin{tikzcd} v_1 & \dots & v_k & v \\ h_1 & \dots \arrow[l] & h_k \arrow[l] & x \arrow[l] \end{tikzcd}$$

Proof by induction:

- **Induction base ($v_1 - 1 \leq h_1$):** (follows from Case I or II)
- **Induction step ($v_{j-1} - 1 \leq h_{j-1} \Rightarrow v_j - 1 \leq h_j$):** we have:
  - $h_{j-1} + 1 = h_j$ (by definition, minimum is horizontal)
  - $v_{j-1} \leq h_{j-1} + 1$ (by induction assumption)
  - $v_j - 1 \leq v_{j-1}$ (by recurrence relation)

Taken together: $v_j - 1 \leq v_{j-1} \leq h_{j-1} + 1 = h_j$ □

Horizontal Property

Lemma: Horizontal Property

The value difference between any two **horizontally adjacent** cells is at most one, that is, $|T[i,j] - T[i,j-1]| \leq 1$.

Proof

Symmetric to the proof for the vertical property.

Diagonal Property

Lemma: Diagonal Property

The value of **diagonally adjacent** cells is non-decreasing and the value difference is at most one, that is, $0 \leq T[i,j] - T[i-1,j-1] \leq 1$.

Diagonal Property

Lemma: Diagonal Property

The value of **diagonally adjacent** cells is non-decreasing and the value difference is at most one, that is, $0 \leq T[i,j] - T[i-1,j-1] \leq 1$.

Proof idea

Show separately:

- $T[i,j] - T[i-1,j-1] \leq 1 \Leftrightarrow T[i,j] \leq T[i-1,j-1] + 1$
- $0 \leq T[i,j] - T[i-1,j-1] \Leftrightarrow T[i-1,j-1] \leq T[i,j]$

Diagonal Property

Lemma: Diagonal Property

The value of **diagonally adjacent** cells is non-decreasing and the value difference is at most one, that is, $0 \leq T[i,j] - T[i-1,j-1] \leq 1$.

Proof idea

Show separately:

- $T[i,j] - T[i-1,j-1] \leq 1 \Leftrightarrow T[i,j] \leq T[i-1,j-1] + 1$
  - Follows immediately from recurrence
- $0 \leq T[i,j] - T[i-1,j-1] \Leftrightarrow T[i-1,j-1] \leq T[i,j]$
  - Left to be shown.

Diagonal Property: Show $d \leq x$

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[u] \end{tikzcd}$$

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd}
d & v \\
h & x \arrow[u]
\end{tikzcd}$$

$d \leq v+1$ (horizontal property)

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[u] \end{tikzcd}$$

$d \leq v+1$ (horizontal property)

$v+1 = x$ (origin of minimum)

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[u] \end{tikzcd}$$

$d \leq v+1$ (horizontal property)

$v+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[u] \end{tikzcd}$$

$d \leq v+1$ (horizontal property)

$v+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Case 2: Minimum from horizontal neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[l] \end{tikzcd}$$

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[u] \end{tikzcd}$$

$d \leq v+1$ (horizontal property)

$v+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Case 2: Minimum from horizontal neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[l] \end{tikzcd}$$

$d \leq h+1$ (vertical property)

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[u] \end{tikzcd}$$

$d \leq v+1$ (horizontal property)

$v+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Case 2: Minimum from horizontal neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[l] \end{tikzcd}$$

$d \leq h+1$ (vertical property)

$h+1 = x$ (origin of minimum)

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[u] \end{tikzcd}$$

$d \leq v+1$ (horizontal property)

$v+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Case 2: Minimum from horizontal neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[l] \end{tikzcd}$$

$d \leq h+1$ (vertical property)

$h+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[u] \end{tikzcd}$$

$d \leq v+1$ (horizontal property)

$v+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Case 2: Minimum from horizontal neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[l] \end{tikzcd}$$

$d \leq h+1$ (vertical property)

$h+1 = x$ (origin of minimum)

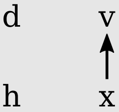
$\longrightarrow d \leq x$ ✓

Case 3: Minimum from diagonal neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[ul] \end{tikzcd}$$

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

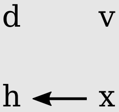


$d \leq v+1$ (horizontal property)

$v+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Case 2: Minimum from horizontal neighbor

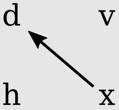


$d \leq h+1$ (vertical property)

$h+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Case 3: Minimum from diagonal neighbor



$x = d$ (match) or $x = d+1$ (mismatch)

Diagonal Property: Show $d \leq x$

Case 1: Minimum from vertical neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[u] \end{tikzcd}$$

$d \leq v+1$ (horizontal property)

$v+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Case 2: Minimum from horizontal neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[l] \end{tikzcd}$$

$d \leq h+1$ (vertical property)

$h+1 = x$ (origin of minimum)

$\longrightarrow d \leq x$ ✓

Case 3: Minimum from diagonal neighbor

$$\begin{tikzcd} d & v \\ h & x \arrow[ul] \end{tikzcd}$$

$x = d$ (match) or $x = d+1$ (mismatch)

$\longrightarrow d \leq x$ ✓

Myers’ Fast Bit Vector Algorithm for Edit Distance

A Fast Bit-Vector Algorithm for Approximate String Matching Based on DP,

Gene Myers, 1999, Journal of the ACM

Bit-parallel computation

Ideas

- We only need to keep one active column in memory
- Consecutive cells in the DP matrix can only differ by {-1,0,1}.
  → We can store differences instead of absolute values.

$$
\begin{array}{c|ccccccc}
 & & \mathtt{B} & \mathtt{A} & \mathtt{N} & \mathtt{A} & \mathtt{N} & \mathtt{A} \\
\hline
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\mathtt{A} & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\
\mathtt{N} & 2 & 2 & 1 & 0 & 1 & 0 & 1 \\
\mathtt{N} & 3 & 3 & 2 & 1 & 1 & 1 & 1 \\
\mathtt{A} & 4 & 4 & 3 & 2 & 1 & 2 & 1
\end{array}
$$

Bit-parallel computation

Definitions

Instead of considering absolute values, we track differences between adjacent rows and columns in the DP matrix:

$$\begin{aligned}
\Delta h_{i,j} &:= T[i,j] - T[i,j-1] \in \{-1,0,1\} && \text{(horizontal)}\\
\Delta v_{i,j} &:= T[i,j] - T[i-1,j] \in \{-1,0,1\} && \text{(vertical)}\\
\Delta d_{i,j} &:= T[i,j] - T[i-1,j-1] \in \{0,1\} && \text{(diagonal)}
\end{aligned}$$

Bit-parallel computation

Definitions

Instead of considering absolute values, we track differences between adjacent rows and columns in the DP matrix:

$$\begin{aligned}
\Delta h_{i,j} &:= T[i,j] - T[i,j-1] \in \{-1,0,1\} && \text{(horizontal)}\\
\Delta v_{i,j} &:= T[i,j] - T[i-1,j] \in \{-1,0,1\} && \text{(vertical)}\\
\Delta d_{i,j} &:= T[i,j] - T[i-1,j-1] \in \{0,1\} && \text{(diagonal)}
\end{aligned}$$

Note

Indices i, j denote the same cell as in the DP matrix.
However, during the algorithm we never have the full matrix with all boolean variables in memory, but only single columns, each of which is stored as a bit vector.

Bit-parallel computation

Original costs from vertical differences

The original values of the full edit distance DP matrix **(left)** can be retrieved from the matrix of $\Delta v_{i,j}$ values **(right)**:

$$T[i,j] = \sum_{r=0}^{i} \Delta v_{r,j}$$

		B	A	N	A	N	A
	0	0	0	0	0	0	0
A	1	1	0	1	0	1	0
N	2	2	1	0	1	0	1
N	3	3	2	1	1	1	1
A	4	4	3	2	1	2	1

		B	A	N	A	N	A
	0	0	0	0	0	0	0
A	+1	+1	0	+1	0	+1	0
N	+1	+1	+1	-1	+1	-1	+1
N	+1	+1	+1	+1	0	+1	0
A	+1	+1	+1	+1	0	+1	0

Bit-parallel computation

Horizontal connections in last row

We can keep track of the relevant last row with $cost_j := T[m, j]$ by using

$$\begin{aligned} cost_0 &= m\,, \\ cost_j &= cost_{j-1} + \Delta h_{m,j}\,. \end{aligned}$$

$$\begin{array}{c|ccccccc} & & \mathtt{B} & \mathtt{A} & \mathtt{N} & \mathtt{A} & \mathtt{N} & \mathtt{A} \\ \hline & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathtt{A} & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ \mathtt{N} & 2 & 2 & 1 & 0 & 1 & 0 & 1 \\ \mathtt{N} & 3 & 3 & 2 & 1 & 1 & 1 & 1 \\ \mathtt{A} & 4 & 4 & 3 & 2 & 1 & 2 & 1 \end{array}$$

$$\begin{array}{c|ccccccc} & & \mathtt{B} & \mathtt{A} & \mathtt{N} & \mathtt{A} & \mathtt{N} & \mathtt{A} \\ \hline & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathtt{A} & +1 & +1 & 0 & +1 & 0 & +1 & 0 \\ \mathtt{N} & +1 & +1 & +1 & -1 & +1 & -1 & +1 \\ \mathtt{N} & +1 & +1 & +1 & +1 & 0 & +1 & 0 \\ \mathtt{A} & +1 & +1 & +1 & +1 & 0 & +1 & 0 \end{array}$$

Bit-parallel computation

Definitions and Encodings

The differences cann be tracked using bit vectors as follows:

$$VP_{i,j} = \begin{cases} 1 & \text{if } \Delta v_{i,j} = 1\,, \\ 0 & \text{otherwise.} \end{cases} \qquad VN_{i,j} = \begin{cases} 1 & \text{if } \Delta v_{i,j} = -1\,, \\ 0 & \text{otherwise.} \end{cases}$$

$$HP_{i,j} = \begin{cases} 1 & \text{if } \Delta h_{i,j} = 1\,, \\ 0 & \text{otherwise.} \end{cases} \qquad HN_{i,j} = \begin{cases} 1 & \text{if } \Delta h_{i,j} = -1\,, \\ 0 & \text{otherwise.} \end{cases}$$

$$D0_{i,j} = \begin{cases} 1 & \text{if } \Delta d_{i,j} = 0\,, \\ 0 & \text{otherwise.} \end{cases}$$

Bit-parallel computation

Definitions and Encodings

The differences cann be tracked using bit vectors as follows:

$$VP_{i,j} = \begin{cases} 1 & \text{if } \Delta v_{i,j} = 1\,, \\ 0 & \text{otherwise.} \end{cases} \qquad VN_{i,j} = \begin{cases} 1 & \text{if } \Delta v_{i,j} = -1\,, \\ 0 & \text{otherwise.} \end{cases}$$

$$HP_{i,j} = \begin{cases} 1 & \text{if } \Delta h_{i,j} = 1\,, \\ 0 & \text{otherwise.} \end{cases} \qquad HN_{i,j} = \begin{cases} 1 & \text{if } \Delta h_{i,j} = -1\,, \\ 0 & \text{otherwise.} \end{cases}$$

$$D0_{i,j} = \begin{cases} 1 & \text{if } \Delta d_{i,j} = 0\,, \\ 0 & \text{otherwise.} \end{cases}$$

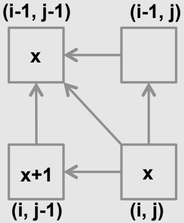
$$\Delta v_{i,j} = VP_{i,j} - VN_{i,j}$$
$$\Delta h_{i,j} = HP_{i,j} - HN_{i,j}$$
$$\Delta d_{i,j} = 1 - D0_{i,j}$$

Dependencies between variables

Observation

$$HN_{i,j} \Leftrightarrow VP_{i,j-1} \text{ and } D0_{i,j}$$

Proof idea



$DO_{i,j}$	$VP_{i,j}$	$HN_{i,j}$
0	0	0
0	1	0
1	0	0
1	1	1

Dependencies between variables

More dependencies

$$HN_{i,j} \Leftrightarrow VP_{i,j-1} \text{ and } D0_{i,j}$$

$$VN_{i,j} \Leftrightarrow HP_{i-1,j} \text{ and } D0_{i,j}$$

$$VP_{i,j} \Leftrightarrow HN_{i-1,j} \text{ or not } (HP_{i-1,j} \text{ and } D0_{i,j})$$

$$HP_{i,j} \Leftrightarrow VN_{i,j-1} \text{ or not } (VP_{i,j-1} \text{ and } D0_{i,j})$$

$$D0_{i,j} \Leftrightarrow (p_i == t_j) \text{ or } VN_{i,j-1} \text{ or } HN_{i-1,j}$$

Dependencies between variables

More dependencies

$$
\begin{aligned}
HN_{i,j} &\Leftrightarrow VP_{i,j-1} \text{ and } D0_{i,j} \\
VN_{i,j} &\Leftrightarrow HP_{i-1,j} \text{ and } D0_{i,j} \\
VP_{i,j} &\Leftrightarrow HN_{i-1,j} \text{ or not } (HP_{i-1,j} \text{ and } D0_{i,j}) \\
HP_{i,j} &\Leftrightarrow VN_{i,j-1} \text{ or not } (VP_{i,j-1} \text{ and } D0_{i,j}) \\
D0_{i,j} &\Leftrightarrow (p_i == t_j) \text{ or } VN_{i,j-1} \text{ or } HN_{i-1,j}
\end{aligned}
$$

Idea of Myers' algorithm

- Store full columns (length m) of each quantity in a bit vector.
- Apply bit-parallel update formulas based on dependencies and text character.

Problem: Circular Dependencies

Problem

For example, $D0$ and HN depend on each other (same j)!

$$HN_{i,j} \Leftrightarrow VP_{i,j-1} \text{ and } D0_{i,j}$$
$$D0_{i,j} \Leftrightarrow (p_i == t_j) \text{ or } VN_{i,j-1} \text{ or } HN_{i-1,j}$$

Problem: Circular Dependencies

Problem

For example, $D0$ and HN depend on each other (same j)!

$$HN_{i,j} \Leftrightarrow VP_{i,j-1} \text{ and } D0_{i,j}$$

$$D0_{i,j} \Leftrightarrow (p_i == t_j) \text{ or } VN_{i,j-1} \text{ or } HN_{i-1,j}$$

Solution

- Denote another bit vector X with $X_j = (p[i] == t[j])$.
- Pre-compute the bit vector for every possible text character $t[j]$.
- The following formula computes the complete bit vector $D0$ from the previous VP:

$$D0 = (((VP \& X) + VP) \wedge VP) \mid X$$

Code: Myers' Algorithm

```python
def myers(P,T,k):
    B = defaultdict(int)
    i = 1
    for c in P:
      B[c] = B[c] | i
      i = i << 1
    m = len(P)
    VP = (1 << m) - 1  # set m bits
    VN = 0
    cost = m
    # ... (next slide)
```

Code: Myers' Algorithm

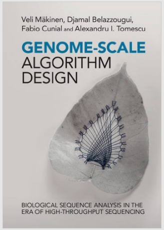
```
def myers(P,T,k):
    # ... preprocessing (previous slide)
    for i, c in enumerate(T):
        Eq = B[c]
        Xv = Eq | VN
        Xh = (((Eq & VP) + VP) ^ VP) | Eq
        HP = VN | ~(Xh|VP)
        HN = VP & Xh
        if HP & (1<<(m-1)):
            cost += 1
        elif HN & (1<<(m-1)):
            cost -= 1
        if cost <= k:
            yield i
        HP = HP << 1
        VP = (HN<<1) | ~(Xv|HP)
        VN = HP & Xv
```

Summary: Myers' Bit-Vector Algorithm

Idea

- Neighboring cells in DP table can differ by at most one
- Encode these differences using bit vectors
- Keep track of score in bottom row
- Formulate algorithm in terms of bit-parallel operations

Complete description of Myers' algorithm


Veli Mäkinen, Djamal Belazzougui,
Fabio Cunial and Alexandru I. Tomescu
GENOME-SCALE
ALGORITHM
DESIGN
BIOLOGICAL SEQUENCE ANALYSIS IN THE
ERA OF HIGH-THROUGHPUT SEQUENCING

Mäkinen, Belazzougui, Cunial, Tomescu
Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing

(see Section 6.1.3 for Myers' algorithm)

Summary

- Variants of the error-tolerant pattern matching problem
- Global vs. semi-global alignment and their edit graphs
- General graph alignment algorithm
- Traceback: follow back edges that led to optima
- Ukkonen's speed-up: $last_k(j+1) \leq last_k(j)+1$
- Proof of differences in DP table: horizontal, vertical, and diagonal properties
- Ideas of Myers' bit-parallel algorithm: difference encoding

Exam Questions

- Define variations of the error tolerant pattern matching problem.
- How does error-tolerant pattern search relate to semi-global alignment?
- How does the edit / alignment graph differ from that for global alignment?
- Explain Ukkonen's speed-up (in theory / on a small example)
- Explain the horizontal, vertical, and diagonal properties.
- What are the ideas to prove these properties?
- Why are these properties helpful?
- What is the idea behind Myers' bit vector algorithm?
- How does the original DP matrix relate to the used bit vectors?