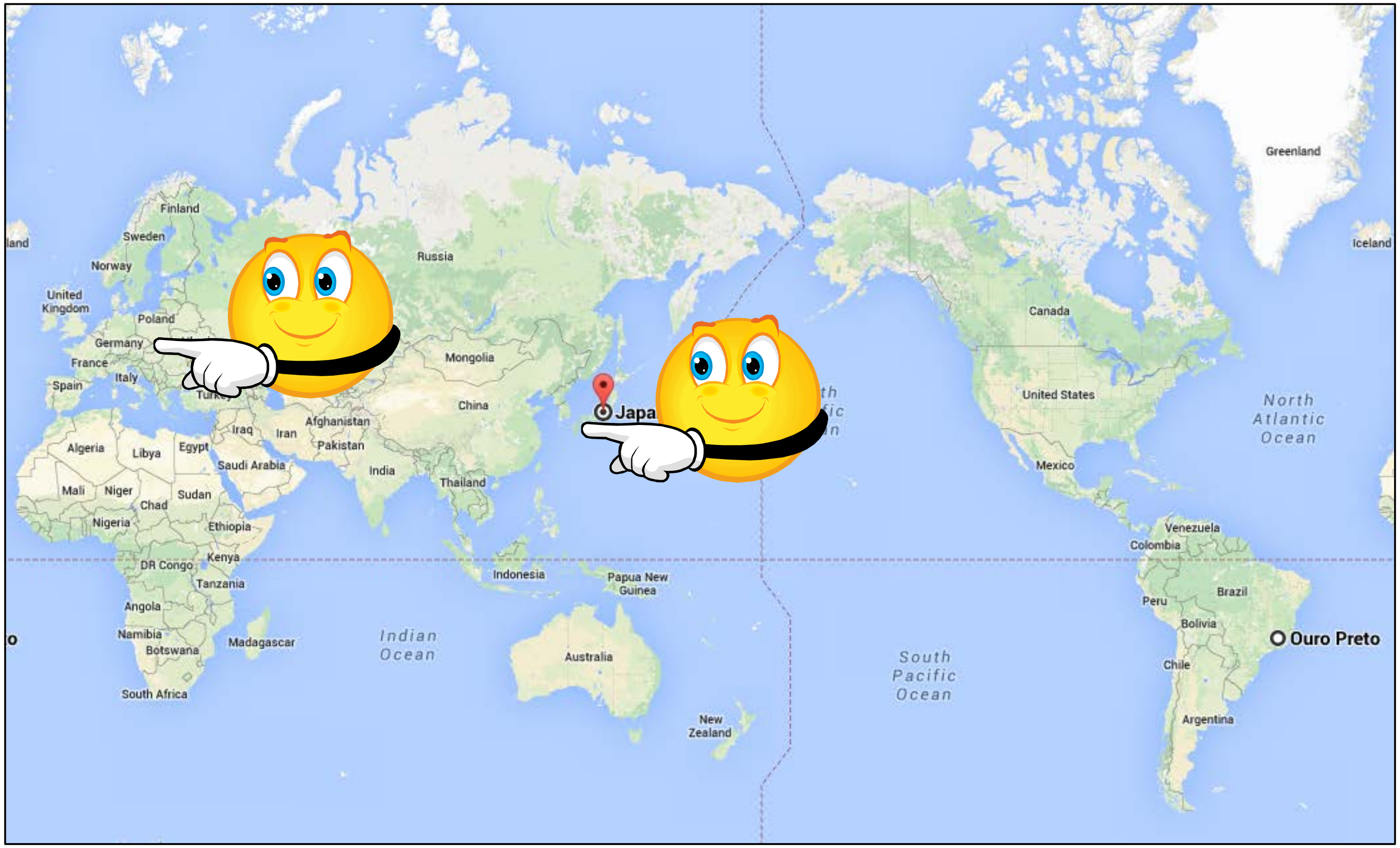
SOFSEM 2016

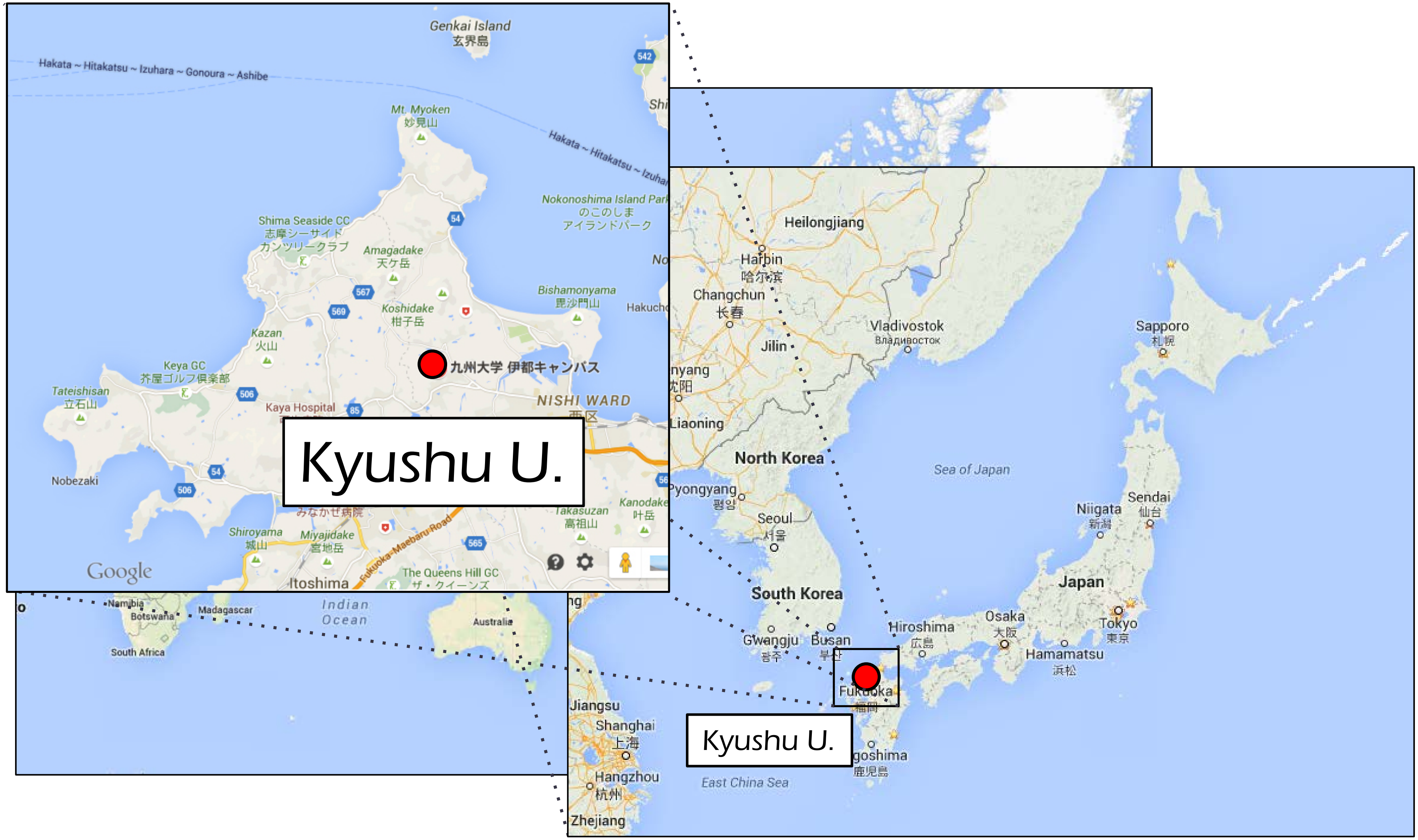
# Compacting a Dynamic Edit Distance Table by RLE Compression

Heikki Hyyrö (University of Tampere, Finland)
Shunsuke Inenaga (Kyushu University, Japan)

# Kyushu University, Japan



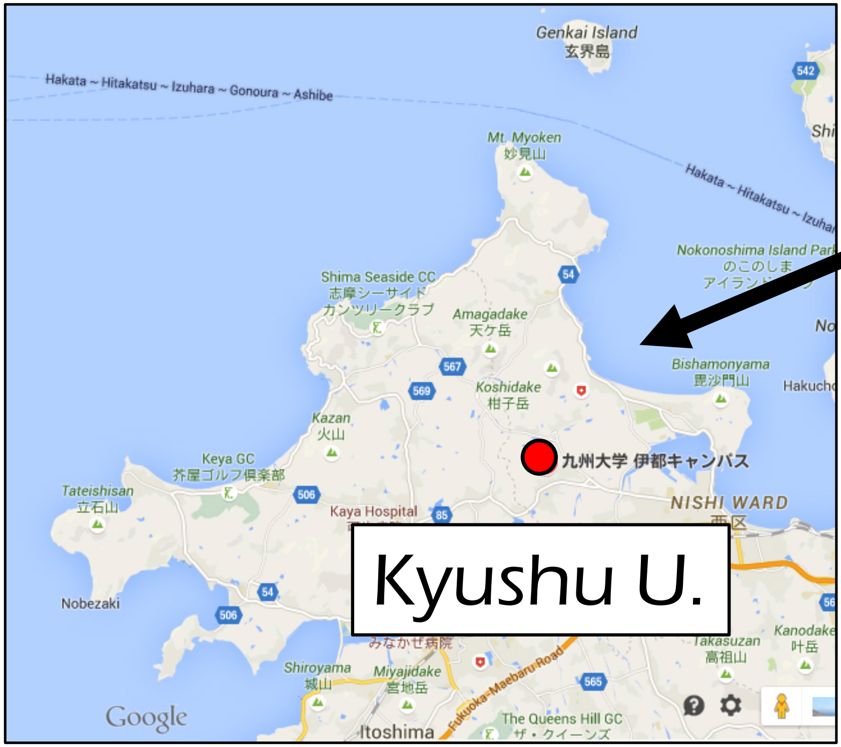
# Kyushu University, Japan



Kyushu U.

Kyushu U.

# Kyushu University, Japan



Kyushu U.

Itoshima Peninsula

糸島

String Island

# String Comparison

Problem 1 (Edit Distance)

Input: two strings $A$ and $B$

Output: the edit distance $ed(A, B)$ between $A$ and $B$

- $ed(A, B)$ is the minimum number of edit operations (insertion, deletion, substitution of a single character) which transforms $A$ to $B$ (or vice versa).

# Dynamic Programming (DP)

- Let $m = |A|$ & $n = |B|$. Let $D$ be a table of size $(m+1) \times (n+1)$ s.t. $D[i, j] = ed(A[1..i], B[1..j])$,
- The fundamental way to compute $D[m, n] = ed(A, B)$ is DP with the following recurrence:
  - $D[i, 0] = i$ for $1 \leq i \leq m$,
  - $D[0, j] = j$ for $1 \leq j \leq n$,
  - $D[i, j] = \min\{ D[i, j\text{-}1]+1, D[i\text{-}1, j]+1,$
    $D[i\text{-}1, j\text{-}1] + \delta(A[i], B[j]) \}$,

    where $\delta(A[i], B[j]) = 1$ if $A[i] \neq B[j]$,
    $\delta(A[i], B[j]) = 0$ if $A[i] = B[j]$.

# Dynamic Programming (DP)

$D$ $B$

$A$

| | | a | t | c | c | g | a | t |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t | 1 | | | | | | | |
| g | 2 | | | | | | | |
| c | 3 | | | | | | | |
| a | 4 | | | | | | | |
| t | 5 | | | | | | | |
| a | 6 | | | | | | | |
| t | 7 | | | | | | | |

$A =$ **tgcatat**
$B =$ **atccgat**

$D[i, 0] = i \text{ for } 1 \leq i \leq m$
$D[0, j] = j \text{ for } 1 \leq j \leq n$

# Dynamic Programming (DP)

*D* *B*

*A*

| | | a | t | c | c | g | a | t |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| g | 2 | 2 | 2 | 2 | 3 | 3 | 4 | 5 |
| c | 3 | 3 | 3 | 2 | 2 | 3 | 4 | 5 |
| a | 4 | 3 | 4 | 3 | 3 | 3 | 3 | 4 |
| t | 5 | 4 | 3 | 4 | 4 | 4 | 4 | 3 |
| a | 6 | 5 | 4 | 4 | 5 | 5 | 4 | |
| t | 7 | 6 | 5 | 5 | 5 | 6 | 5 | |

$A = \texttt{tgcatat}$

$B = \texttt{atccgat}$

$$\begin{aligned} D[i, j] = \min\{ & D[i, j\text{-}1]+1, \\ & D[i\text{-}1, j]+1, \\ & D[i\text{-}1, j\text{-}1] +1\} \end{aligned}$$

# Dynamic Programming (DP)

$D$ $B$

$A$

|   |   | **a** | **t** | **c** | **c** | **g** | **a** | **t** |
|---|---|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **t** | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| **g** | 2 | 2 | 2 | 2 | 3 | 3 | 4 | 5 |
| **c** | 3 | 3 | 3 | 2 | 2 | 3 | 4 | 5 |
| **a** | 4 | 3 | 4 | 3 | 3 | 3 | 3 | 4 |
| **t** | 5 | 4 | 3 | 4 | 4 | 4 | 4 | 3 |
| **a** | 6 | 5 | 4 | 4 | 5 | 5 | 4 | 4 |
| **t** | 7 | 6 | 5 | 5 | 5 | 6 | 5 |   |

$A$ = **`tgcatat`**

$B$ = **`atccgat`**

$$\begin{aligned} D[i, j] = \min\{ &D[i, j\text{-}1]+1, \\ &\boxed{D[i\text{-}1, j]+1}, \\ &D[i\text{-}1, j\text{-}1] +1\} \end{aligned}$$

# Dynamic Programming (DP)

$D$ $B$

$A$

| | | **a** | **t** | **c** | **c** | **g** | **a** | **t** |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **t** | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| **g** | 2 | 2 | 2 | 2 | 3 | 3 | 4 | 5 |
| **c** | 3 | 3 | 3 | 2 | 2 | 3 | 4 | 5 |
| **a** | 4 | 3 | 4 | 3 | 3 | 3 | 3 | 4 |
| **t** | 5 | 4 | 3 | 4 | 4 | 4 | 4 | 3 |
| **a** | 6 | 5 | 4 | 4 | 5 | 5 | 4 | 4 |
| **t** | 7 | 6 | 5 | 5 | 5 | 6 | 5 | |

$A$ = **`tgcatat`**
$B$ = **`atccgat`**

$$D[i, j] = \min\{\ D[i, j\text{-}1]+1,\ D[i\text{-}1, j]+1,\ D[i\text{-}1, j\text{-}1]\ \}$$

# Dynamic Programming (DP)

$D$ $B$

$A$

| | | a | t | c | c | g | a | t |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| t | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| g | 2 | 2 | 2 | 2 | 3 | 3 | 4 | 5 |
| c | 3 | 3 | 3 | 2 | 2 | 3 | 4 | 5 |
| a | 4 | 3 | 4 | 3 | 3 | 3 | 3 | 4 |
| t | 5 | 4 | 3 | 4 | 4 | 4 | 4 | 3 |
| a | 6 | 5 | 4 | 4 | 5 | 5 | 4 | 4 |
| t | 7 | 6 | 5 | 5 | 5 | 6 | 5 | 4 |

$A =$ **`tgcatat`**

$B =$ **`atccgat`**

$$D[i, j] = \min\{\begin{aligned} & D[i, j\text{-}1]+1, \\ & D[i\text{-}1, j]+1, \\ & \boxed{D[i\text{-}1, j\text{-}1]}\ \} \end{aligned}$$

$O(mn)$ total time

# Cyclic Rotation of String

- For $1 \leq j \leq n$, let $B_j = B[j..n]B[1..j\text{-}1]$, i.e., $B_j$ is the $j$-th cyclic rotation of $B$.
- E.g.) If $B$ = **SOFSEM**, then
  - $B_1$ = **SOFSEM**
  - $B_2$ = **OFSEMS**
  - $B_3$ = **FSEMSO**
  - $B_4$ = **SEMSOF**
  - $B_5$ = **EMSOFS**
  - $B_6$ = **MSOFSE**

# Cyclic String Comparison

**Problem 2 (Cyclic Edit Distance)**

Input: two strings $A$ and $B$

Output: the edit distance $ed(A, B_j)$ for $A$ and all rotations $B_1, \ldots, B_n$ of $B$.

- Motivation in bioinformatics (some biological sequences are circular).
- Naïve approach takes $O(mn)$ time for each rotation $B_j$. So, overall it takes $O(mn^2)$ time.
- Any better solution?

# Right Increment Is Easy

$B[1..5]$

$A$

|   |   | c | a | g | t | a |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| a | 1 | 1 | 1 | 2 | 3 | 4 |
| g | 2 | 2 | 2 | 1 | 2 | 3 |
| c | 3 | 2 | 3 | 2 | 2 | 3 |
| t | 4 | 3 | 3 | 3 | 2 | 3 |
| a | 5 | 4 | 3 | 4 | 3 | 2 |

$B[1..5]B[1]$

$A$

|   |   | c | a | g | t | a | c |
|---|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| a | 1 | 1 | 1 | 2 | 3 | 4 | 5 |
| g | 2 | 2 | 2 | 1 | 2 | 3 | 4 |
| c | 3 | 2 | 3 | 2 | 2 | 3 | 3 |
| t | 4 | 3 | 3 | 3 | 2 | 3 | 4 |
| a | 5 | 4 | 3 | 4 | 3 | 2 | 3 |

- New values are only at the last column.
  ⇒ Right increment takes $O(m)$ time.

# Left Decrement Is NOT as Easy

$B[1..5]B[1]$

$A$

| | | c | a | g | t | a | c |
|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| a | 1 | 1 | 1 | 2 | 3 | 4 | 5 |
| g | 2 | 2 | 2 | 1 | 2 | 3 | 4 |
| c | 3 | 2 | 3 | 2 | 2 | 3 | 3 |
| t | 4 | 3 | 3 | 3 | 2 | 3 | 4 |
| a | 5 | 4 | 3 | 4 | 3 | 2 | 3 |

$B[2..5]B[1]$

$A$

| | | | a | g | t | a | c |
|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 |
| a | | 1 | 0 | 1 | 2 | 3 | 4 |
| g | | 2 | 1 | 0 | 1 | 2 | 3 |
| c | | 3 | 2 | 1 | 1 | 2 | 2 |
| t | | 4 | 3 | 2 | 1 | 2 | 3 |
| a | | 5 | 4 | 3 | 2 | 1 | 2 |

- When the left-most character is deleted, different values can propagate to all columns!

# Algorithms for Left Decrements

| Algorithms | Left decr. time | Space |
|---|---|---|
| Landau et al. (1998) | $O(m + n)$ | $O(mn)$ |
| Schmidt (1998) | $O(m + n)$ | $O(mn)$ |
| Kim & Park (2004) | $O(m + n)$ | $O(mn)$ |
| Hyyrö et al. (2015) | $O(m + n)$ | $O(mn)$ |

- There are several known solutions for the left-decrement edit distance problem.
- Each solution uses some “indirect” representation of the DP table which requires $O(mn)$ space. This space consumption is a bottle neck.

# Run Length Encoding (RLE)

- The RLE of a string $A$ is a compressed representation of $A$ where each maximal "run" $a \ldots a$ of the same character is encoded by $a^p$, where $p$ is the length of the run.
  - E.g.) $RLE$(aaabbcccccbb) = $a^3b^2c^5b^2$
- The size $k$ of $RLE(A)$ is the number of maximal runs in $A$.
- If $m$ is the length of the original string $A$, then clearly $k \leq m$ holds.

# DR Tables (Kim & Park 2004)

- Let $DR$ be a differential representation of DP table $D$ for $ed(A, B)$ such that:
  - $DR[i, j].U = D[i, j] - D[i-1, j]$ (vertical diff.)
  - $DR[i, j].L = D[i, j] - D[i, j-1]$ (horizontal diff.)

$D$

| | | c | a | g | t |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |
| a | 1 | 1 | 1 | 2 | 3 |
| g | 2 | 2 | 2 | 1 | 2 |
| c | 3 | 2 | 3 | 2 | 2 |
| t | 4 | 3 | 3 | 3 | 2 |

$DR.U$

| | | c | a | g | t |
|---|---|---|---|---|---|
| | | | | | |
| a | 1 | 0 | -1 | -1 | -1 |
| g | 1 | 1 | 1 | -1 | -1 |
| c | 1 | 0 | 1 | 1 | 0 |
| t | 1 | 1 | 0 | 1 | 0 |

$DR.L$

| | | c | a | g | t |
|---|---|---|---|---|---|
| | | 1 | 1 | 1 | 1 |
| a | | 0 | 0 | 1 | 1 |
| g | | 0 | 0 | -1 | 1 |
| c | | -1 | 1 | -1 | 0 |
| t | | -1 | 0 | 0 | -1 |

# Property of DR Tables

- Let $DR$ and $DR'$ denote the DR tables for $ed(A, B)$ and $ed(A, B[2..n])$, respectively.

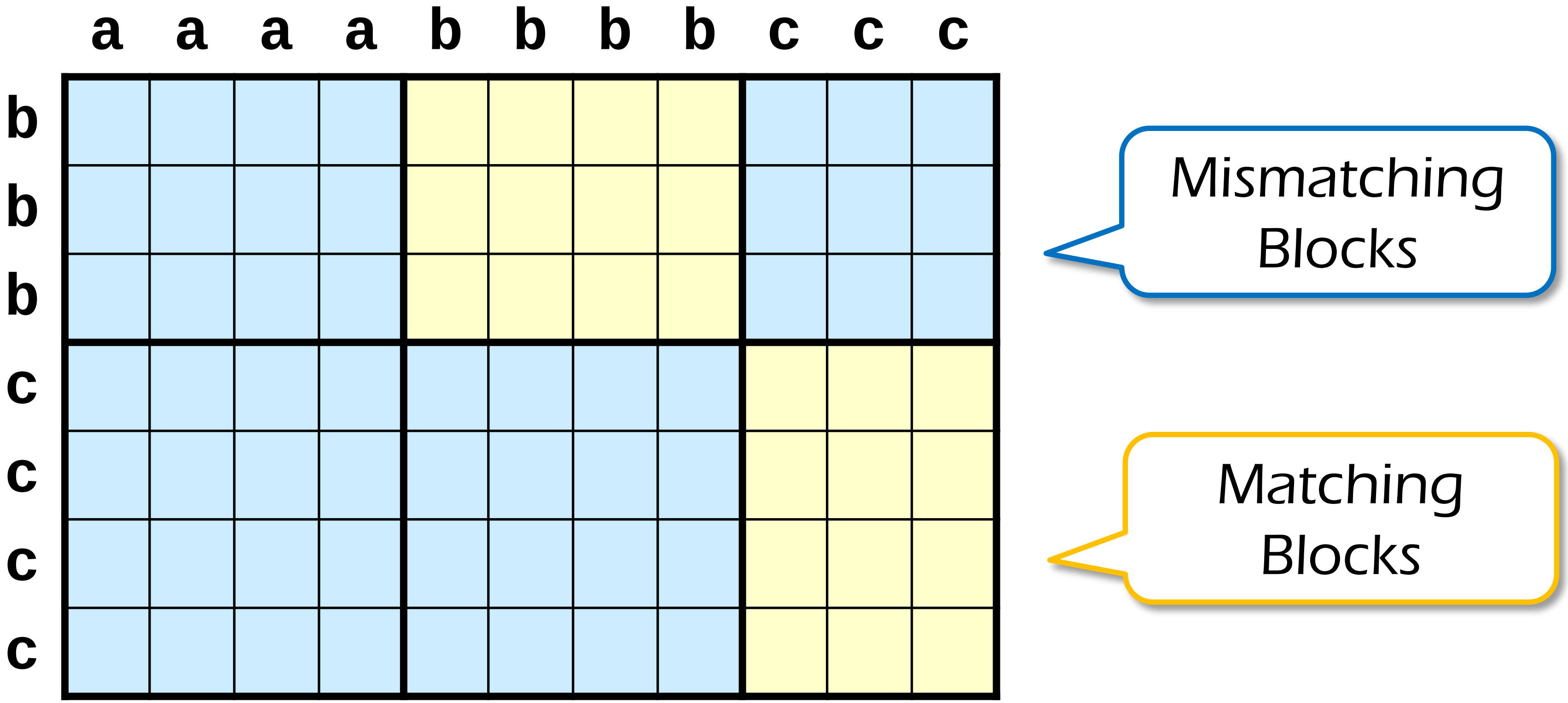
**Theorem 1** [Hyyrö et al. 2015]

For each row $i$ of $DR'$, there are only $O(1)$ column indices $j$ s.t. $DR'[i, j].L \neq DR[i, j].L$ .

For each column $j$ of $DR'$, there are only $O(1)$ row indices $i$ s.t. $DR'[i, j].U \neq DR[i, j].U$ .

# Edit Distance of RLE strings

- The DP and DR tables of $ed(RLE(A), RLE(B))$ can be divided into $kl$ blocks [Arbel et al. 2002].

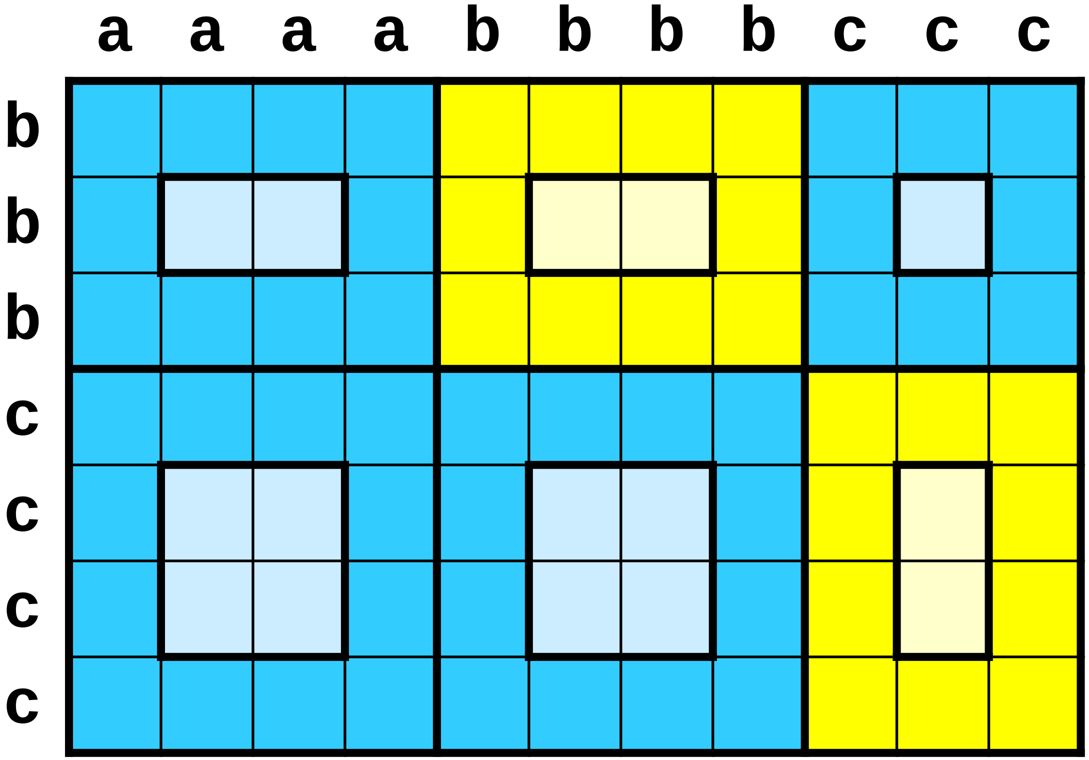


Mismatching Blocks

Matching Blocks

# Edit Distance of RLE strings

- We explicitly store only the block boundaries of the DR tables, using $O(ml + nk)$ space.
- Then, the values inside the blocks can be computed on the fly.



Total number of cells in block boundaries are $O(ml + nk)$.

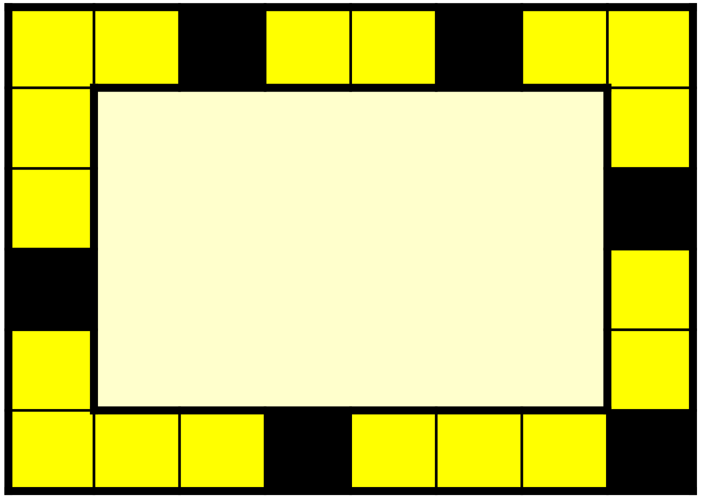
# Key Lemma

**Lemma 1**

Each of the top, bottom, left, and right boundaries of a block of $DR$ contains only $O(1)$ cells $(i, j)$ such that $DR'[i, j] \neq DR[i, j]$.

Proof.

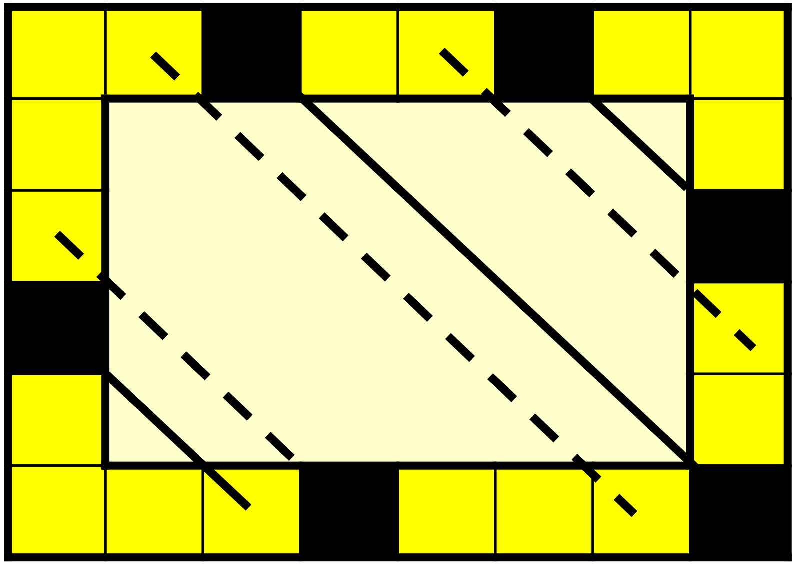
- By Theorem 1.



Black cells are those where $DR'[i, j] \neq DR[i, j]$.

# Processing Matching Blocks

- In a matching block, the values in the DP tables *D*’ and *D* propagate diagonally.
- Thus, the different values of *DR* propagate only diagonally, from left/top boundaries to bottom/right boundaries.



# Processing Matching Blocks

Lemma 2

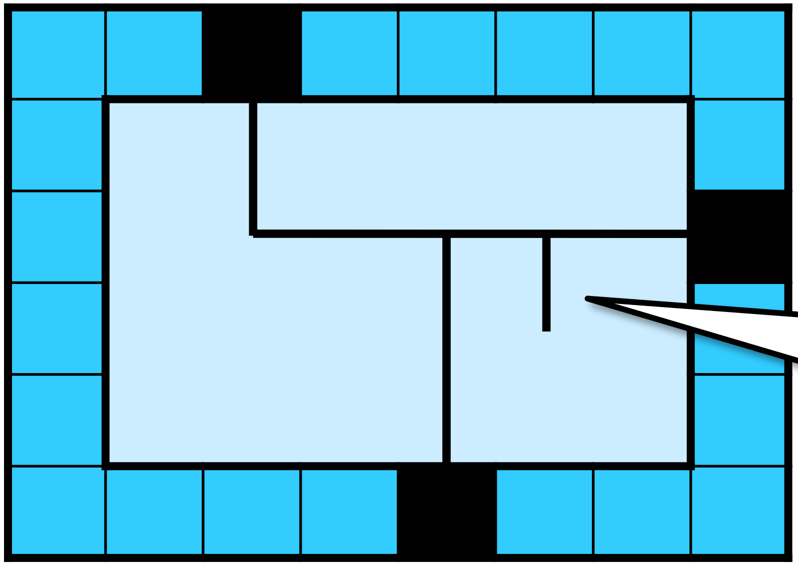
After the left-most character of $B$ is deleted, all matching blocks of the DR table can be updated in a total of $O(m + n)$ time, using $O(ml + nk)$ space.

Proof.

- Moving one step forward in a diagonal path takes $O(1)$ time.
- The total length of diagonal paths in all matching blocks is $O(m + n)$.

# Processing Mismatching Blocks

- In a mismatching block, the different values of *DR’* may diverge.
- From each of the $O(1)$ sources in the left/top boundaries, we trace all paths by DFS.



Some path may not reach the right or bottom boundary.

# Processing Mismatching Blocks

**Lemma 3**

After the left-most character of $B$ is deleted, all mismatching blocks of the DR table can be updated in a total of $O(m + n)$ time, using $O(ml + nk)$ space.

Proof.

- We can traverse all the paths of DFS in time linear in the total length of the paths. (Details are omitted.)

# Processing Mismatching Blocks

**Lemma 3**

After the left-most character of $B$ is deleted, all mismatching blocks of the DR table can be updated in a total of $O(m + n)$ time, using $O(ml + nk)$ space.

Proof. (Cont.)

- The total length of the paths is linear in the number of cells where $DR'[i, j] \neq DR[i, j]$.
- It follows from Theorem 1 that there are only $O(m + n)$ such cells in total.

# Putting All Together

**Theorem 2 (Main result)**

Given an $O(ml + nk)$-space representation of the DR table for $ed(A, B)$, we can update it to that for $ed(A, B[2..n])$ in $O(m + n)$ time.

- $m = |A|$
- $n = |B|$
- $k = |RLE(A)|$
- $l = |RLE(B)|$

# Conclusions and Future Work

- We proposed the first space-efficient left-decremental edit distance algorithm, which is based on RLE.
- Our algorithm can also be applied to the left-incremental case.
- Open questions: Can we extend our algorithm to:
  - Weighted edit distance?
  - Insertion and deletion at arbitrary positions?