SPIRE 2020

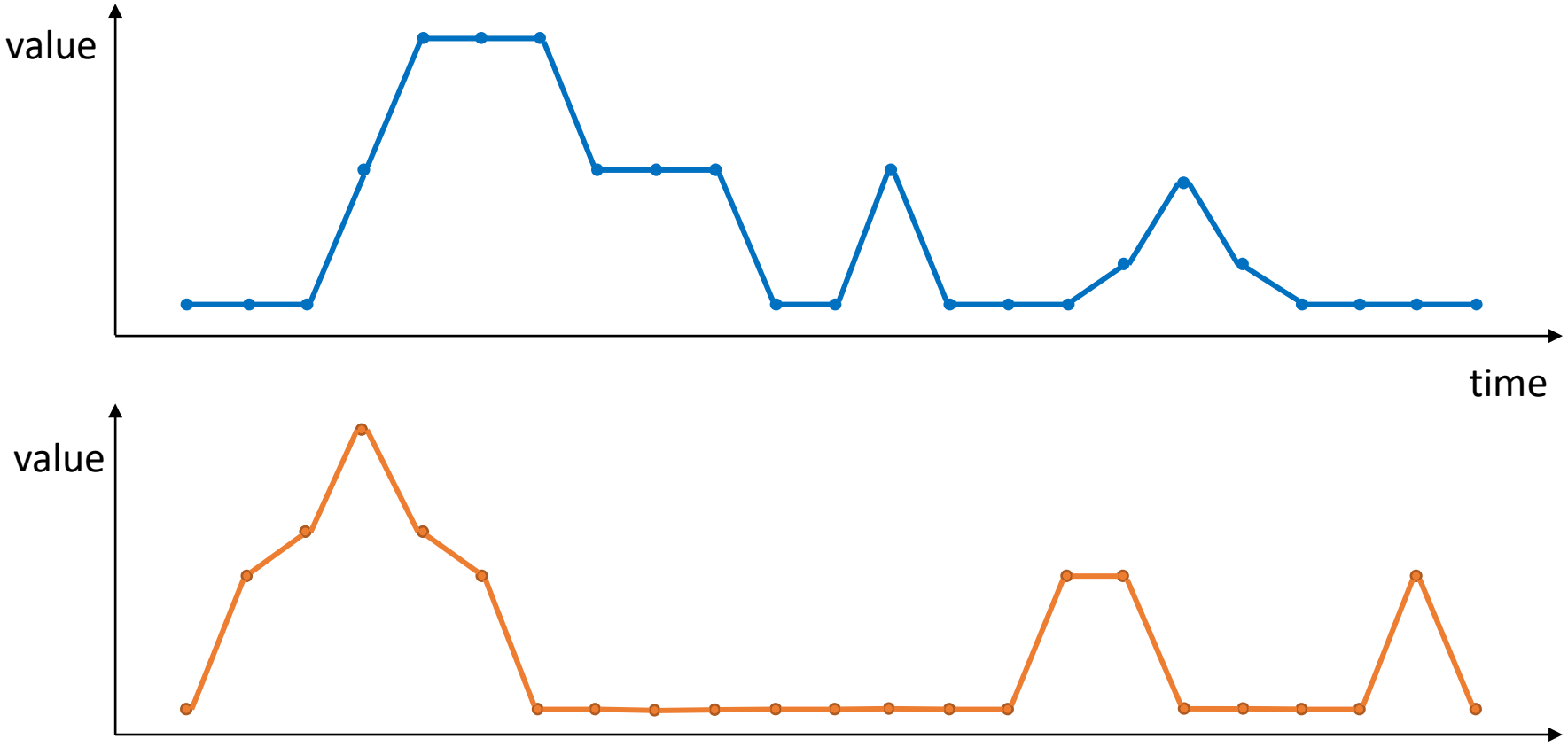
Towards Efficient Interactive Computation of Dynamic Time Warping Distance

Akihiro Nishi[1], Yuto Nakashima[1],
Shunsuke Inenaga[1], Hideo Bannai[2], Masayuki Takeda[1]

[1] Kyushu University
[2] Tokyo Medical and Dental University

Time Series Comparisons

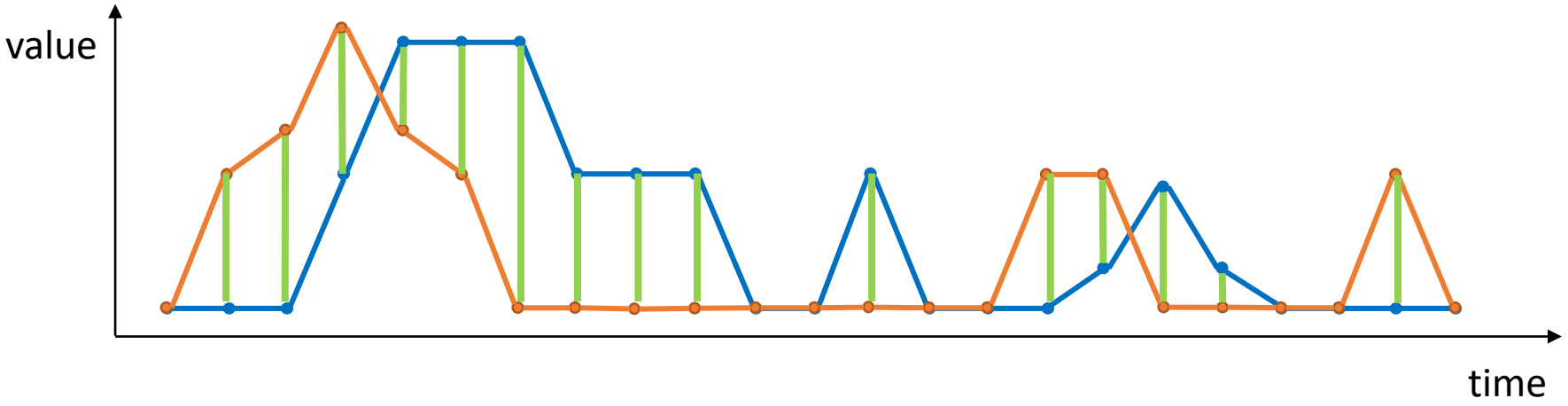
Comparing time series is a central task in data mining. For example, consider the two time series shown below, both having one high peak and two lower peaks following.



Time Series Comparisons [cont.]

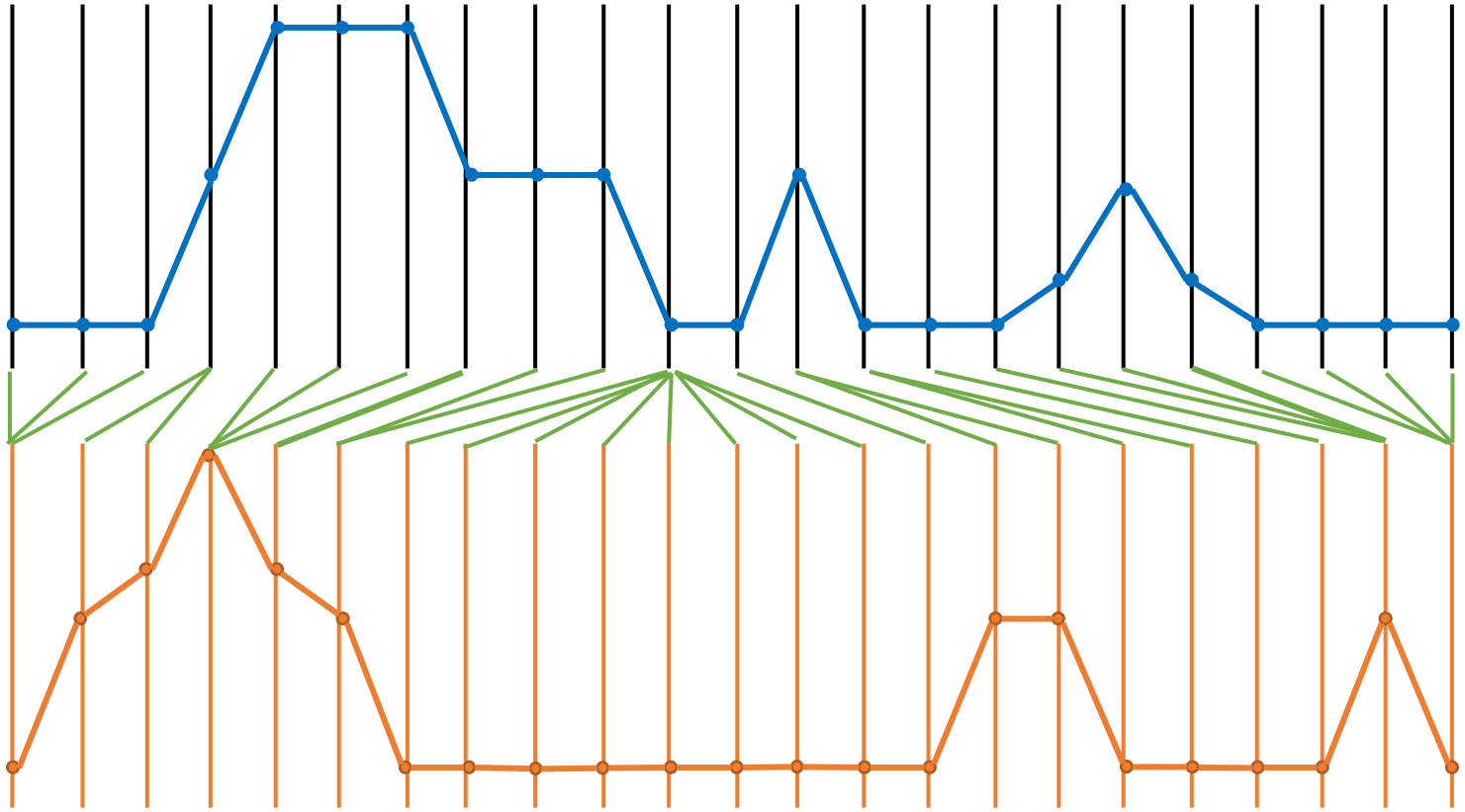
If we use a linear one-to-one mapping, we fail to capture the similarity between these two time series.

This can be resolved by considering a suitable non-linear one-to-many mapping, which will be shown next.



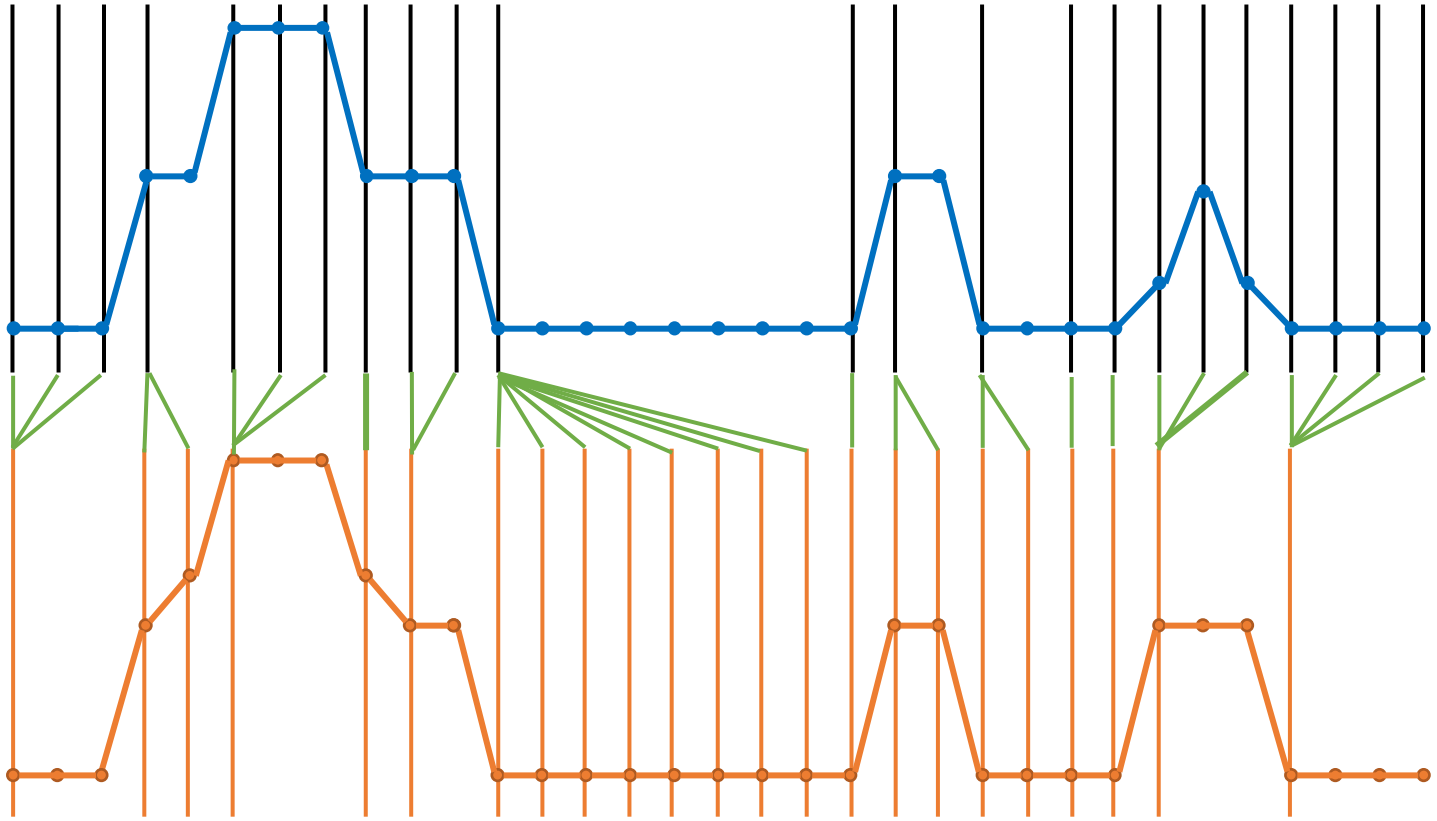
Time Series Comparisons [cont.]

Let’s consider the following one-to-many alignment.

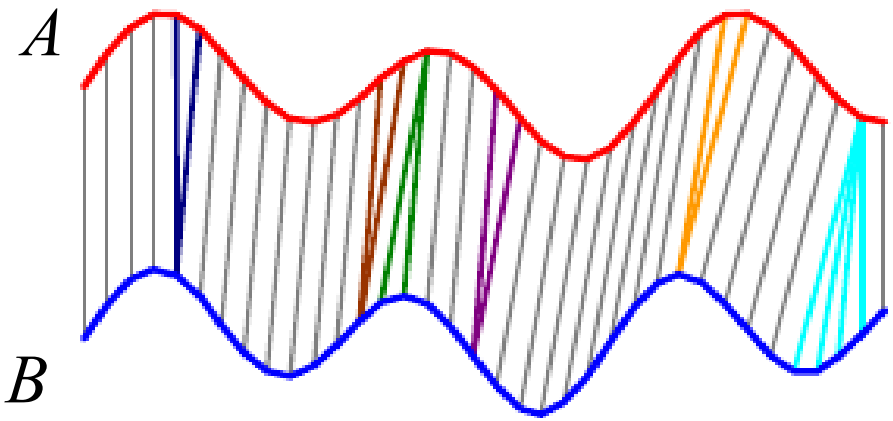


Time Series Comparisons [cont.]

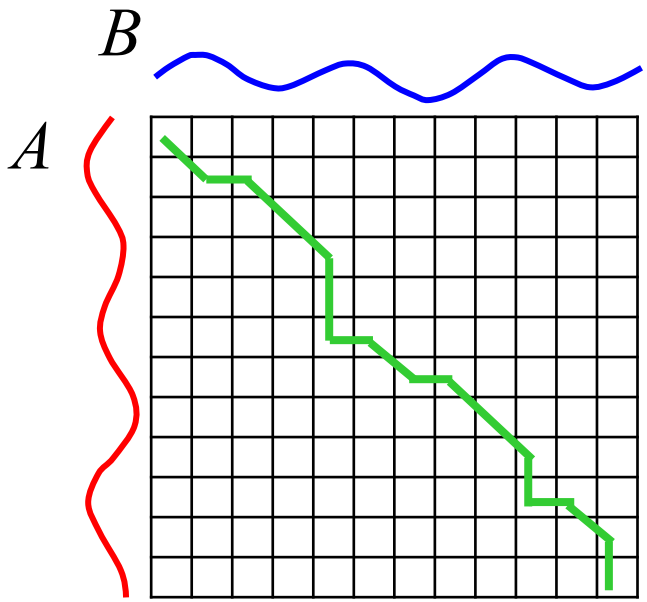
Based on the alignment, we can dynamically adjust the time points so the two time series look very similar. This is exactly what ***Dynamic Time Warping*** does.



Dynamic Time Warping (DTW)



DTW [Sakoe & Chiba 1978]
one-to-many alignment for A and B



DP table for $\mathrm{dtw}(A, B)$

$$\mathrm{dtw}(A, B) = \min_{p \in \mathcal{P}} \sum_{(i,j) \in p} c(a_i, b_j)$$

- $\mathcal{P}$ is the set of all paths in the grid graph from top-left to bottom-right.
- $c(a, b) = |a - b|$ is one of the most commonly used score functions.

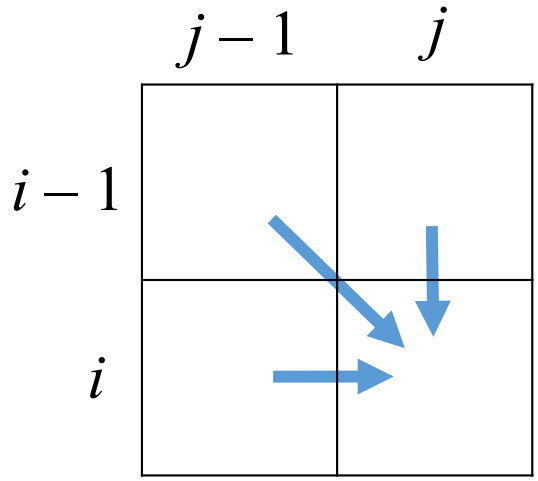
Dynamic Time Warping [cont.]

Let D be an $m \times n$ DP table s.t. $D[i, j] = \mathrm{dtw}(A[1..i], B[1..j])$.

Each $D[i, j]$ can be computed by the following recurrence:

$$D[i, j] = \min\{D[i-1, j], D[i, j-1], D[i-1, j-1]\} + |a_i - b_j|$$

DP table D



$\mathrm{dtw}(A, B)$ can be computed in $\Theta(mn)$ time.

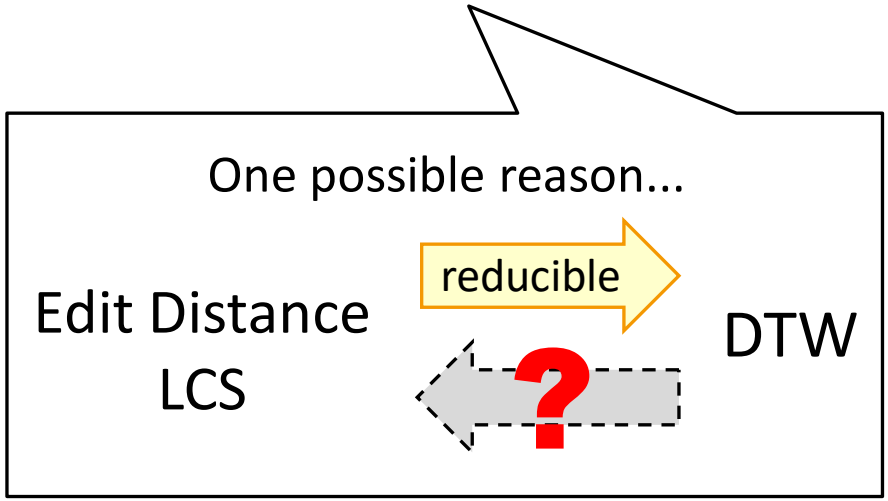
No $O((mn)^{1-\varepsilon})$ DTW algorithm exists unless SETH fails.

Dynamic Time Warping [cont.]

Dynamic Time Warping has a wide range of applications including robotics, biometrics, bioinformatics, image processing, musical information processing, etc.

Sakoe & Chiba's DTW paper has **> 6,000 citations** alone.

However, algorithmic work for computing exact DTW scores has been **very limited**.



[This Work]
First DTW algorithm which allows for edits on the input data.

D^2TW: DTW in Dynamic Setting

Problem (Dynamic Dynamic Time Warping, D^2TW)

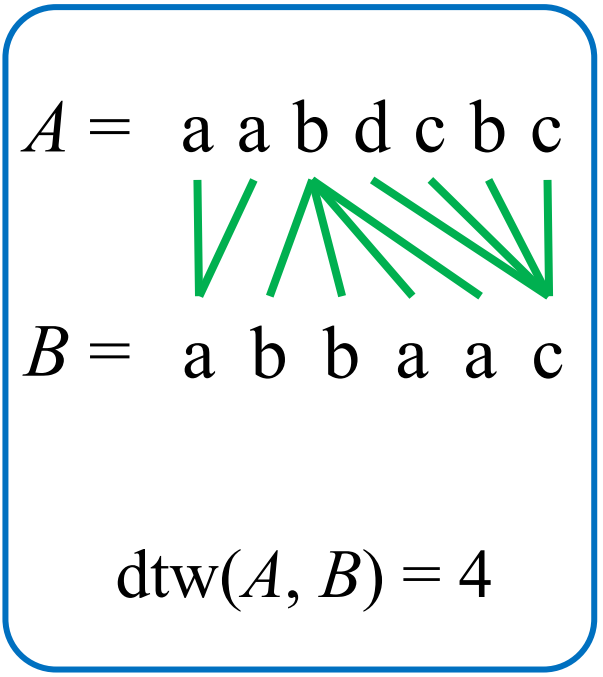
Given

- two strings A and B to preprocess,
- an online sequence of single-character edit operations (insertion/deletion/substitution) on A or B,

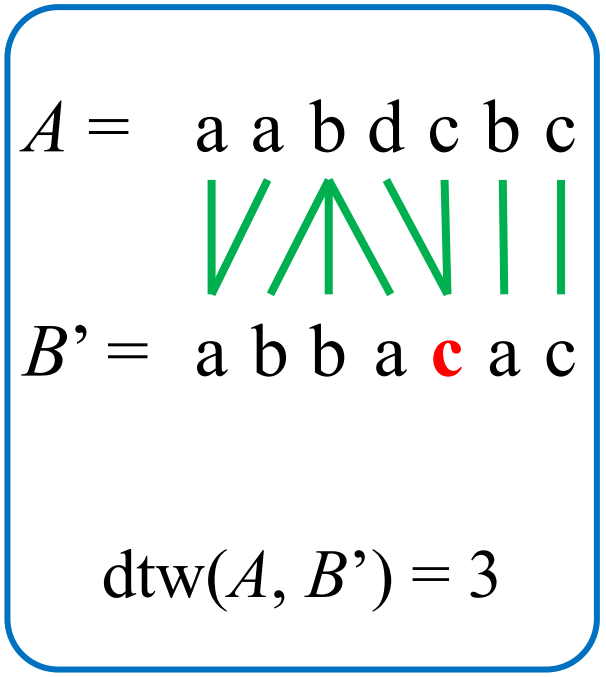
maintain a data structure which stores the DTW distance between the modified strings after each edit operation.

D²TW: DTW in Dynamic Setting [cont.]

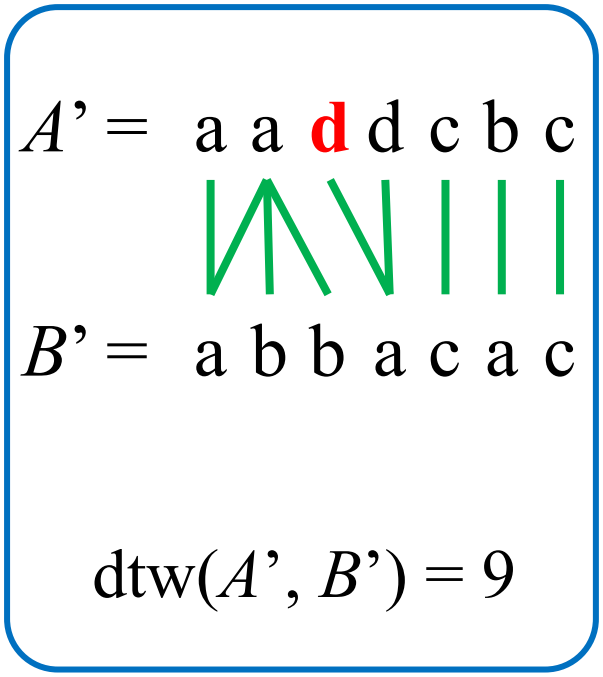
preprocess A and B.



insert letter **c** after position 4 of B.



substitute **b** at pos. 3 in A with **d**.



How Much Does DP Table Change?

In the worst case, $\Theta(mn)$ cells can be different between DP table D before the edit and DP table D' after the edit, where $m = |A|$ and $n = |B|$. ➔ Worst-case $\Theta(mn)$ work per edit.

delete $B[1] = \text{a}$

D

		a	c	b	e	e	a	a	d
		0	1	2	3	4	5	6	7
d	1	3	4	6	7	8	11	14	14
c	2	5	3	4	6	8	10	12	13
b	3	6	4	3	6	9	9	10	12
b	4	7	5	3	6	9	10	10	12
c	5	9	5	4	5	7	9	12	11
c	6	11	5	5	6	7	9	11	12
d	7	14	6	7	6	7	10	12	11
a	8	14	8	7	10	10	7	7	10

D'

		c	b	e	e	a	a	d
		0	1	2	3	4	5	6
d	1	1	3	4	5	8	11	11
c	2	1	2	4	6	7	9	10
b	3	2	1	4	7	7	8	10
b	4	3	1	4	7	8	8	10
c	5	3	2	3	5	7	9	9
c	6	3	3	4	5	7	9	9
d	7	4	5	4	5	8	10	9
a	8	6	5	7	8	5	5	8

Our Approach to D²TW

To achieve a better solution to D²TW, we use two key tools:

1. **Difference representation** DR of the DP table D;
2. **Sparse representation** DS for DR which is based on the run length encoding (**RLE**) of the input strings.

Theorem (D²TW algorithm)

There is a $\Theta(km+ln)$-space data structure for D²TW which can be updated in $O(m + n + \#)$ time per edit, where $m = |A|$, $n = |B|$, $l = |\mathrm{RLE}(A)|$, $k = |\mathrm{RLE}(B)|$, and $\#$ is the number of cells in DS that change after the edit.

➔ $\# = \Omega(km+ln)$ but it could be smaller in many cases.

Previous and This Work for D²TW

Algorithm	Update Time	Space
Naïve DP	$\Theta(mn)$	$\Theta(mn)$
Froese et al.	$\Theta(km+ln)$	$\Theta(km+ln)$
Ours	$\mathrm{O}(m+n+\#)$, where $\# = \mathrm{O}(km+ln)$.	$\Theta(km+ln)$

$m = |A|,\ n = |B|,\ l = |\mathrm{RLE}(A)|,\ k = |\mathrm{RLE}(B)|$

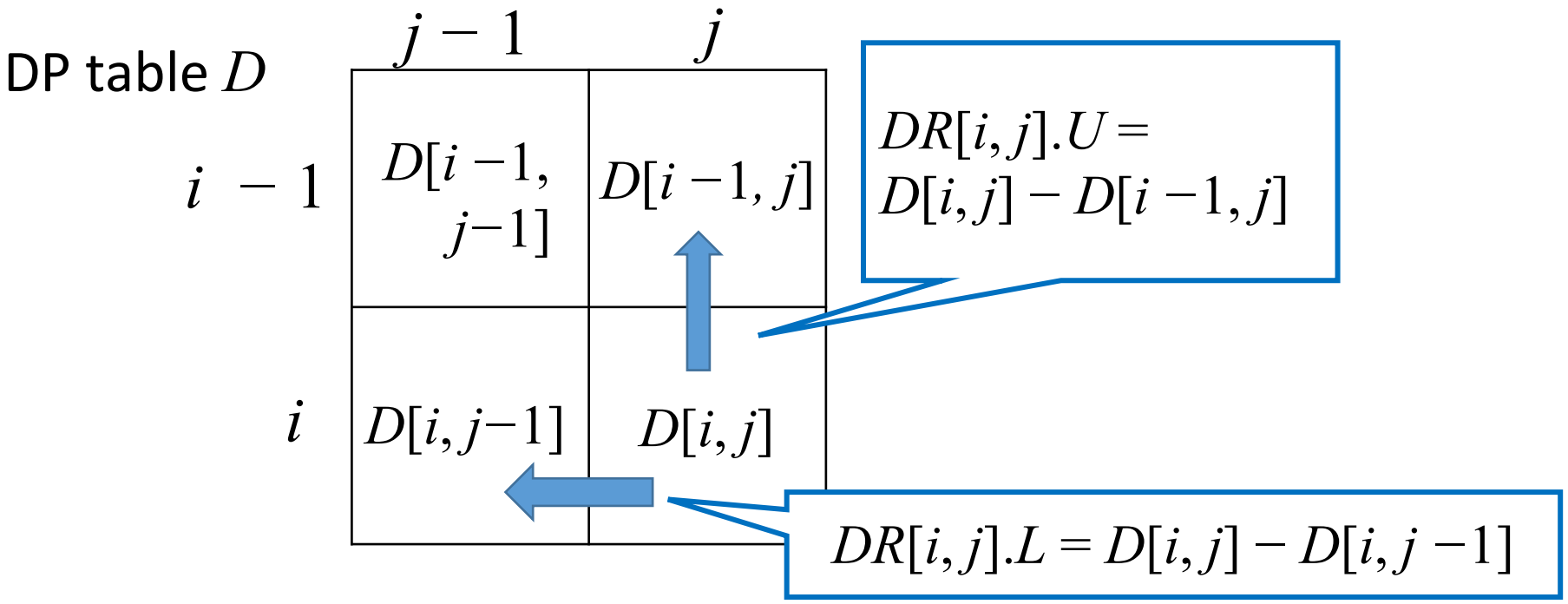
Our method can be regarded as an opportunistic algorithm which can be efficient for some inputs.

In our preliminary experiments with random strings, $\#$ is much smaller than $km + ln$ (to be shown later).

Difference Representation DR

We use a **difference representation** DR of the DP table D, first proposed for edit distance [Kim & Park, 2004].

Each cell $DR[i, j]$ stores the vertical and horizontal differences $DR[i, j].U$ and $DR[i, j].L$ from the neighboring cells.



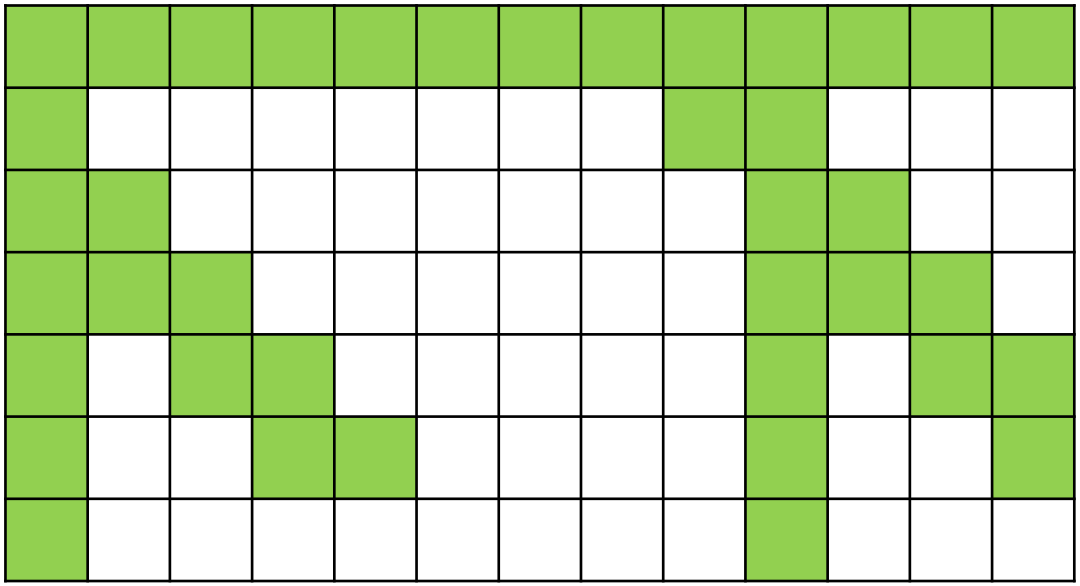
Tracing Update Paths in DR

Let B' be the string B after the edit operation,
DR' the difference representation of DP table D' for A and B'.

For edit distance, Hyyrö & Inenaga showed how to trace the cells in DR' that need to be updated from DR.

We adapt their approach to DTW.

Updating DR to DR'

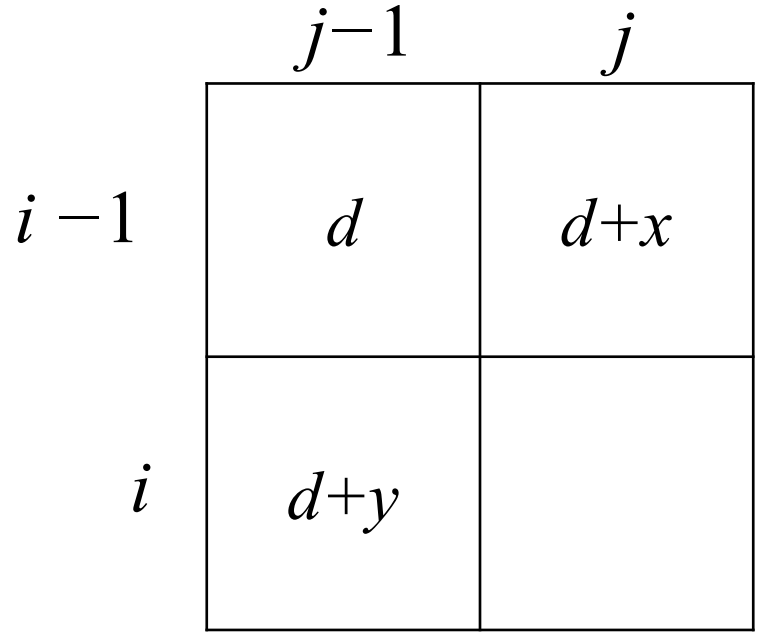


cells that change

cells that do not change

How to Compute DR without D

DP table D
(imaginable)



Let $d = D[i-1, j-1]$,

$$x = D[i-1, j] - D[i-1, j-1] = DR[i-1, j].L\ ,$$

$$y = D[i, j-1] - D[i-1, j-1] = DR[i, j-1].U.$$

How to Compute DR without D [cont.]

DP table D (imaginable)

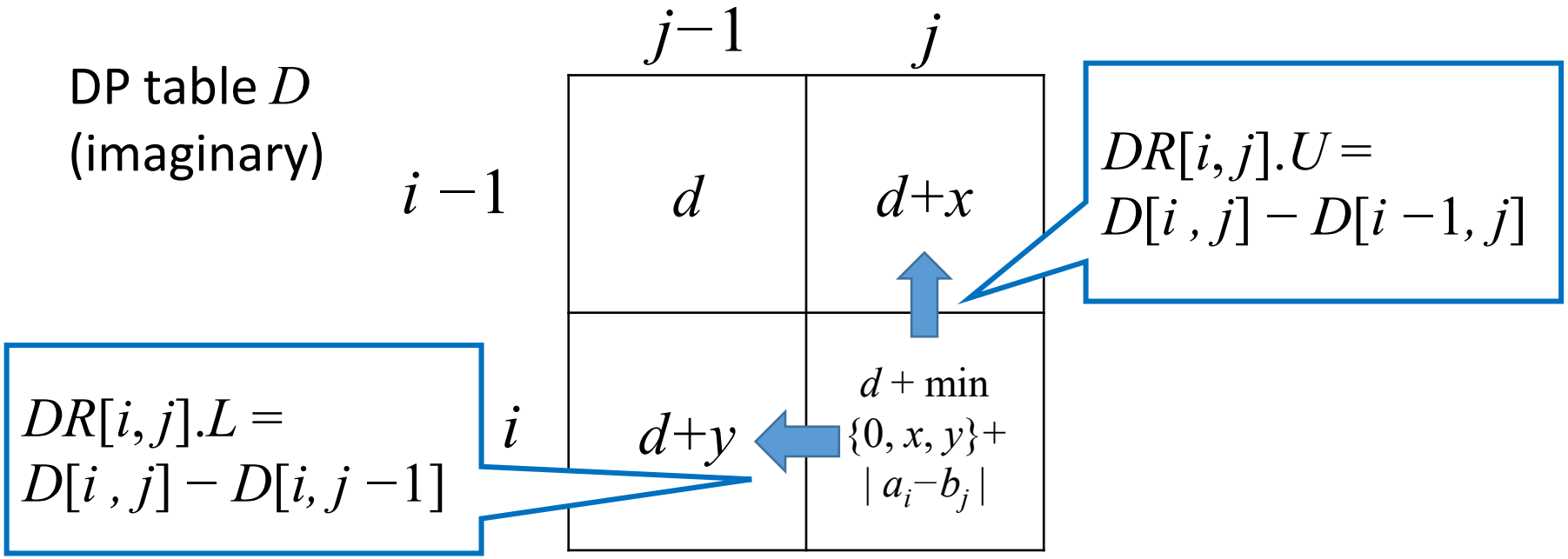
	$j-1$	j
$i-1$	d	$d+x$
i	$d+y$	$d + \min\{0, x, y\} + \lvert a_i - b_j \rvert$

By the recurrence of DTW, we have

$$D[i,j] = \min\{d, d+x, d+y\} + |a_i - b_j|$$
$$= d + \min\{0, x, y\} + |a_i - b_j|$$

How to Compute DR without D [cont.]

DP table D
(imaginary)



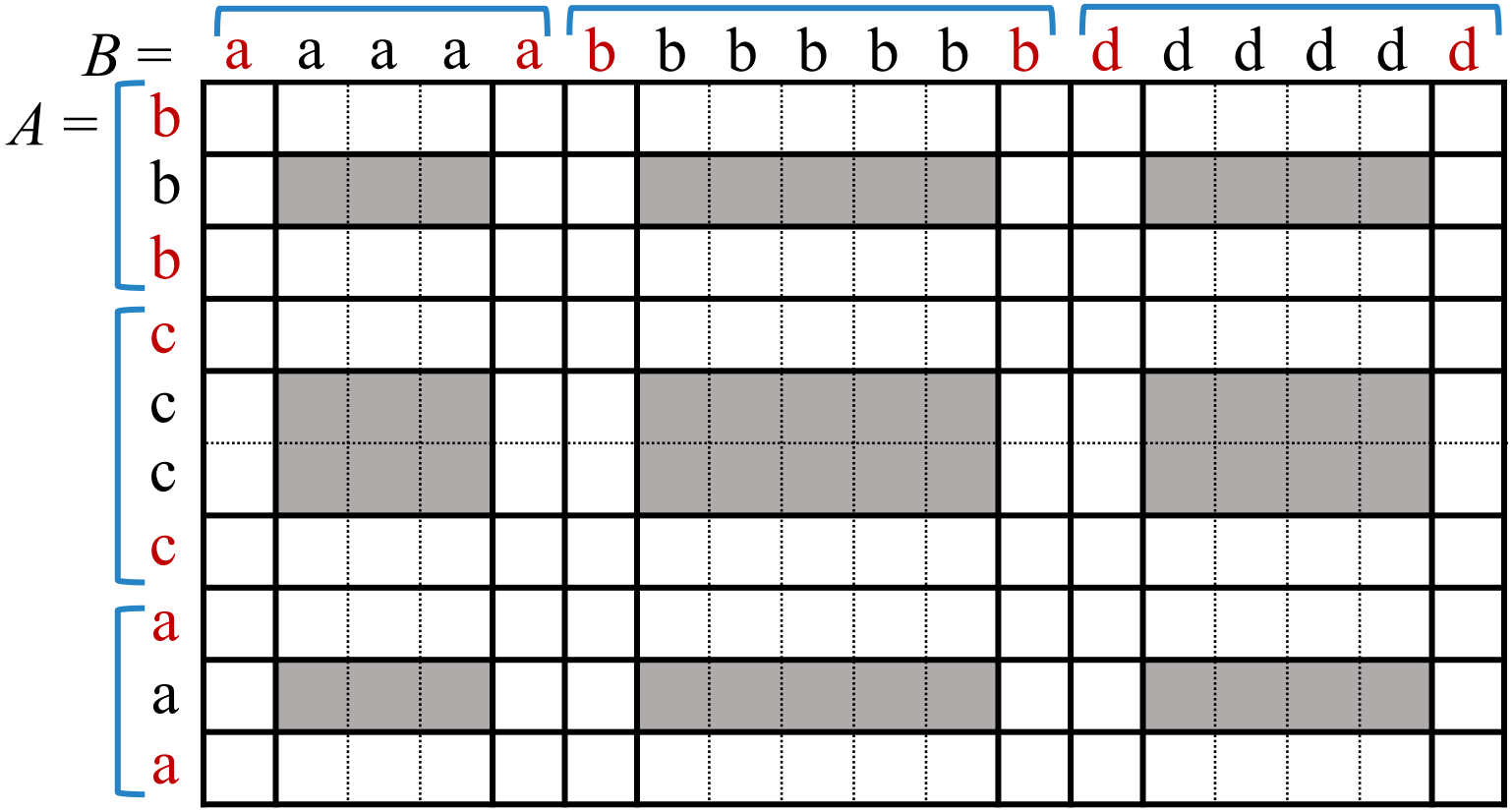
By the definition of DR, we get

- $DR[i,j].U = \min\{0, x, y\} + |a_i - b_j| - x$
- $DR[i,j].L = \min\{0, x, y\} + |a_i - b_j| - y$

➔ We can compute $DR[i,j]$ from the L-field of the upper-neighbor and the U-field of the left-neighbor.

Sparse Representation DS for DR

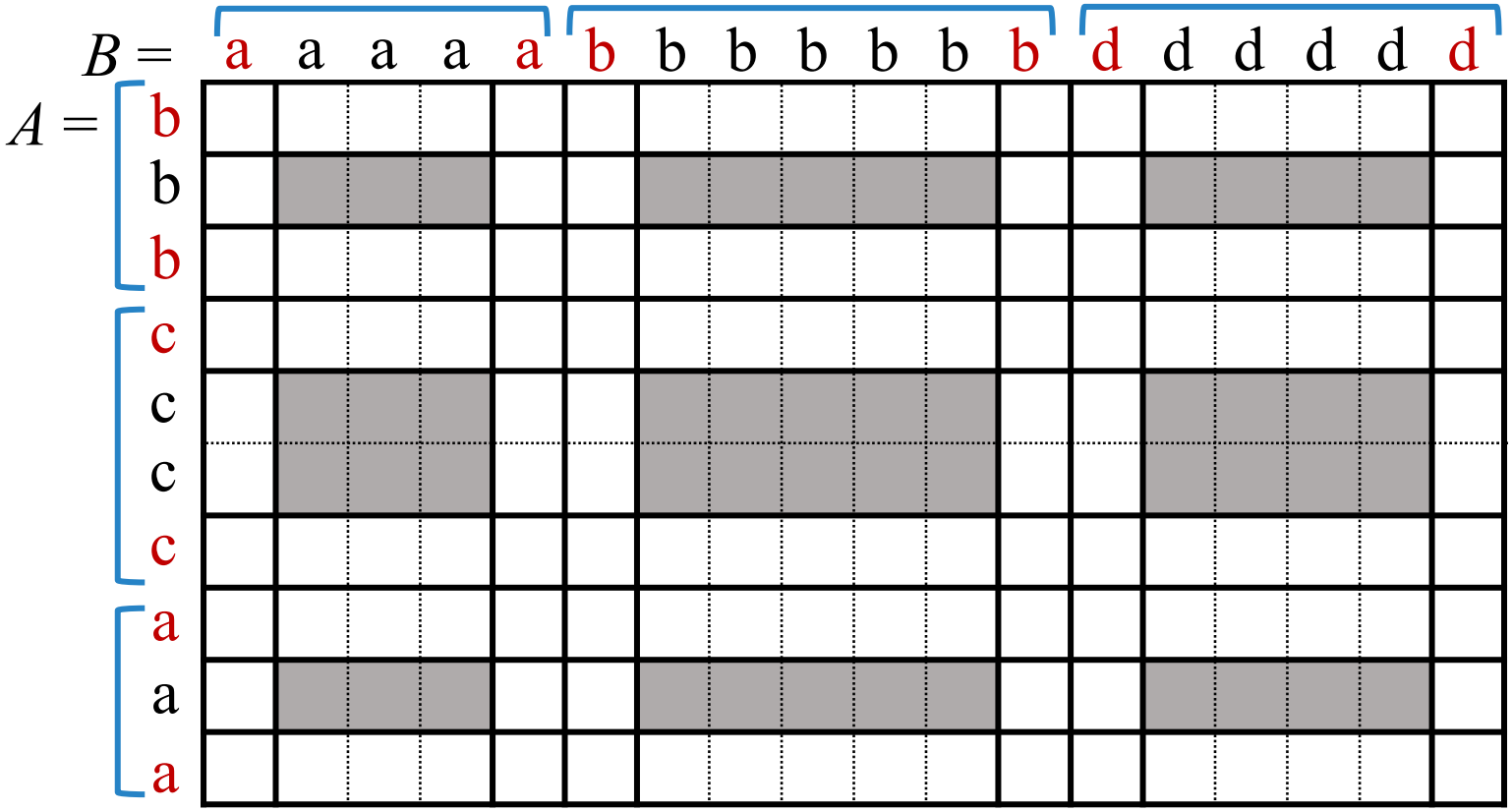
Let DS be a **sparse table** for DR which stores only the rows and columns corresponding to the first and last characters of each character run.



Sparse Representation DS for DR [cont.]

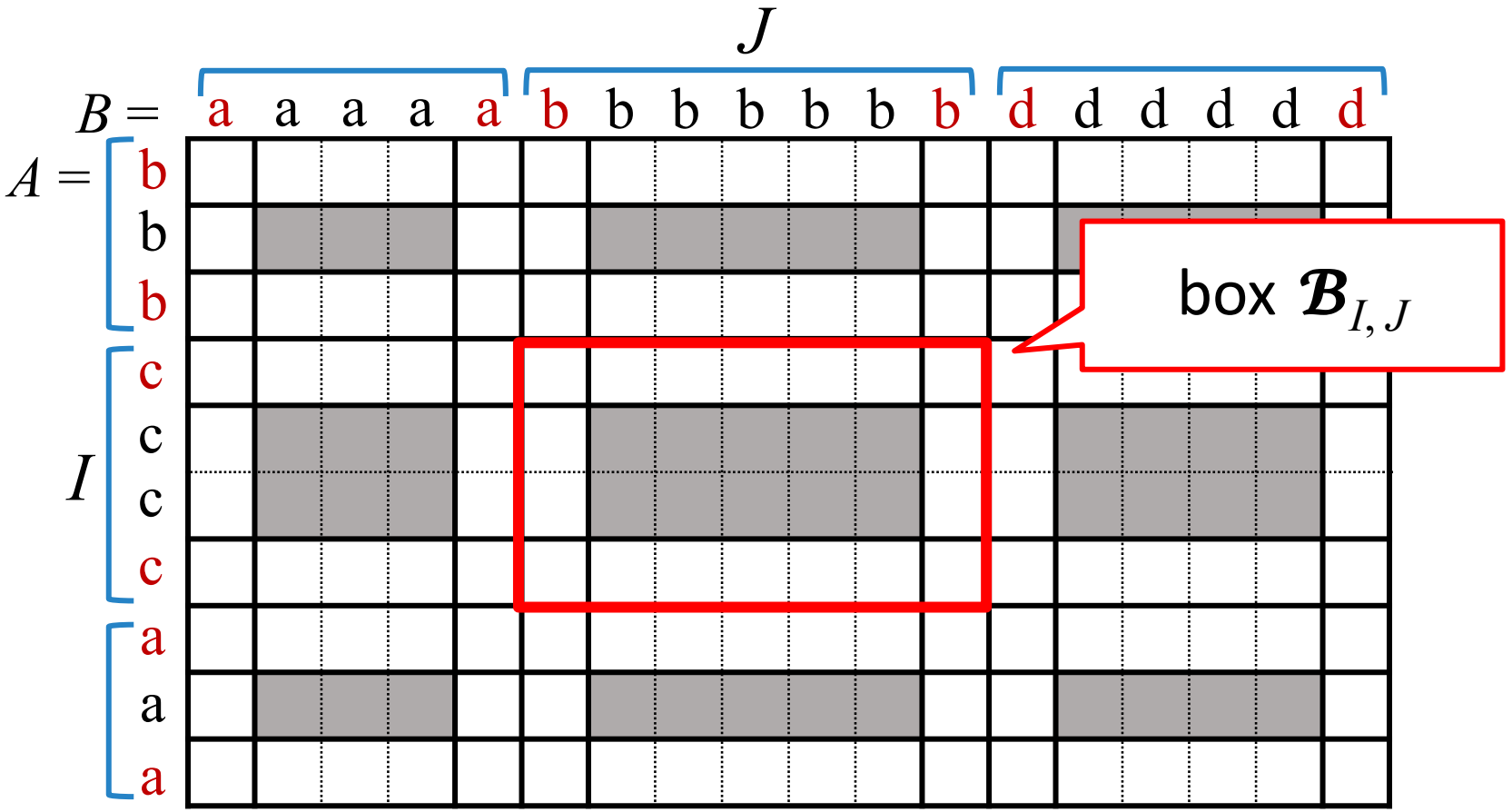
Since we only store the boundary (white) cells, sparse difference table DS requires $\Theta(km + ln)$ space.

$m = |A|$, $n = |B|$, $l = |\mathrm{RLE}(A)|$, $k = |\mathrm{RLE}(B)|$.



Sparse Representation DS for DR [cont.]

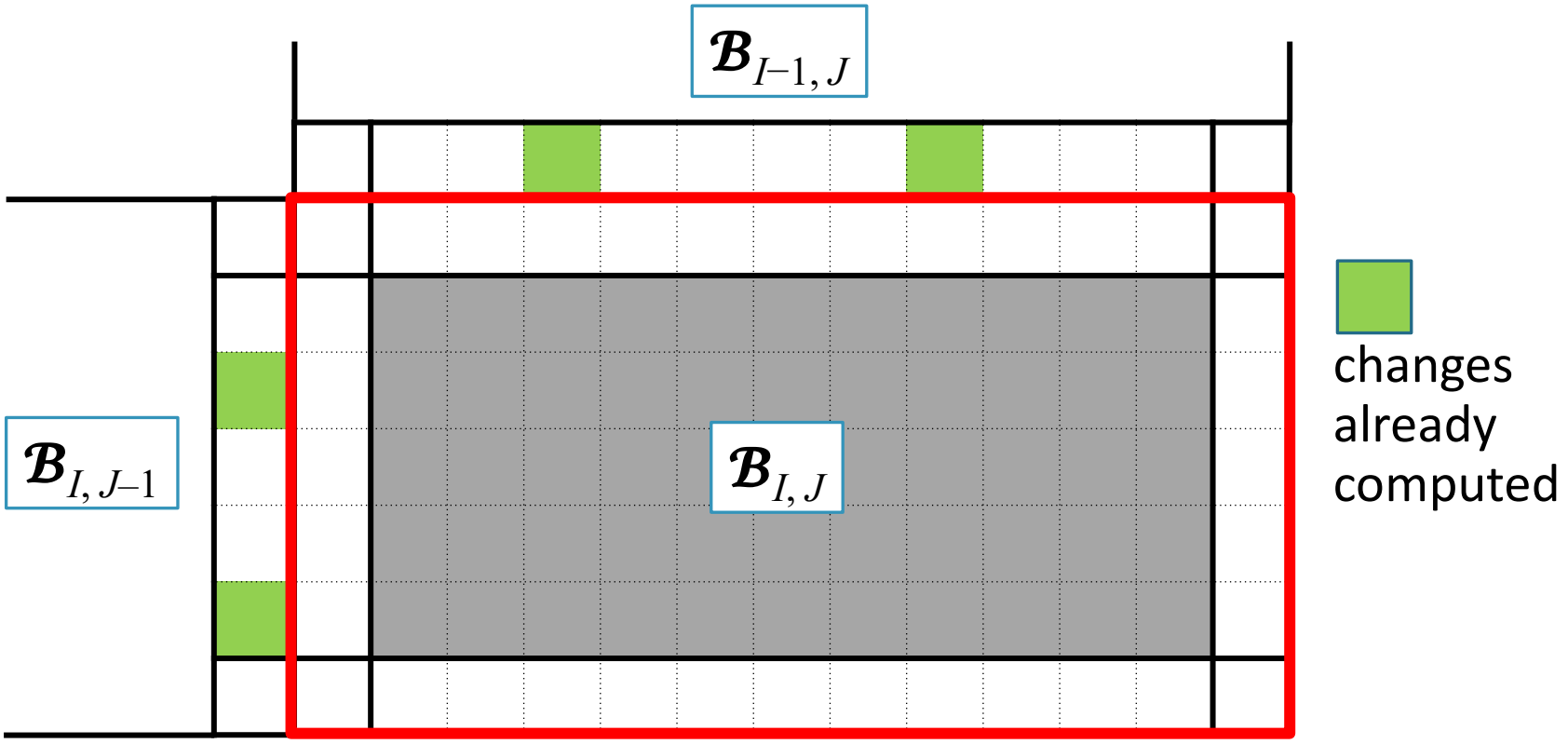
Let $\mathcal{B}_{I,J}$ be the region of DR which is surrounded by the I-th character run of A and the J-th character run of B.



Updating Boundaries of $\mathcal{B}_{I,J}$

Assume that boxes $\mathcal{B}_{I-1,J}$ and $\mathcal{B}_{I,J-1}$ have been processed and we know the cells that were changed in $\mathcal{B}_{I-1,J}$ and $\mathcal{B}_{I,J-1}$.

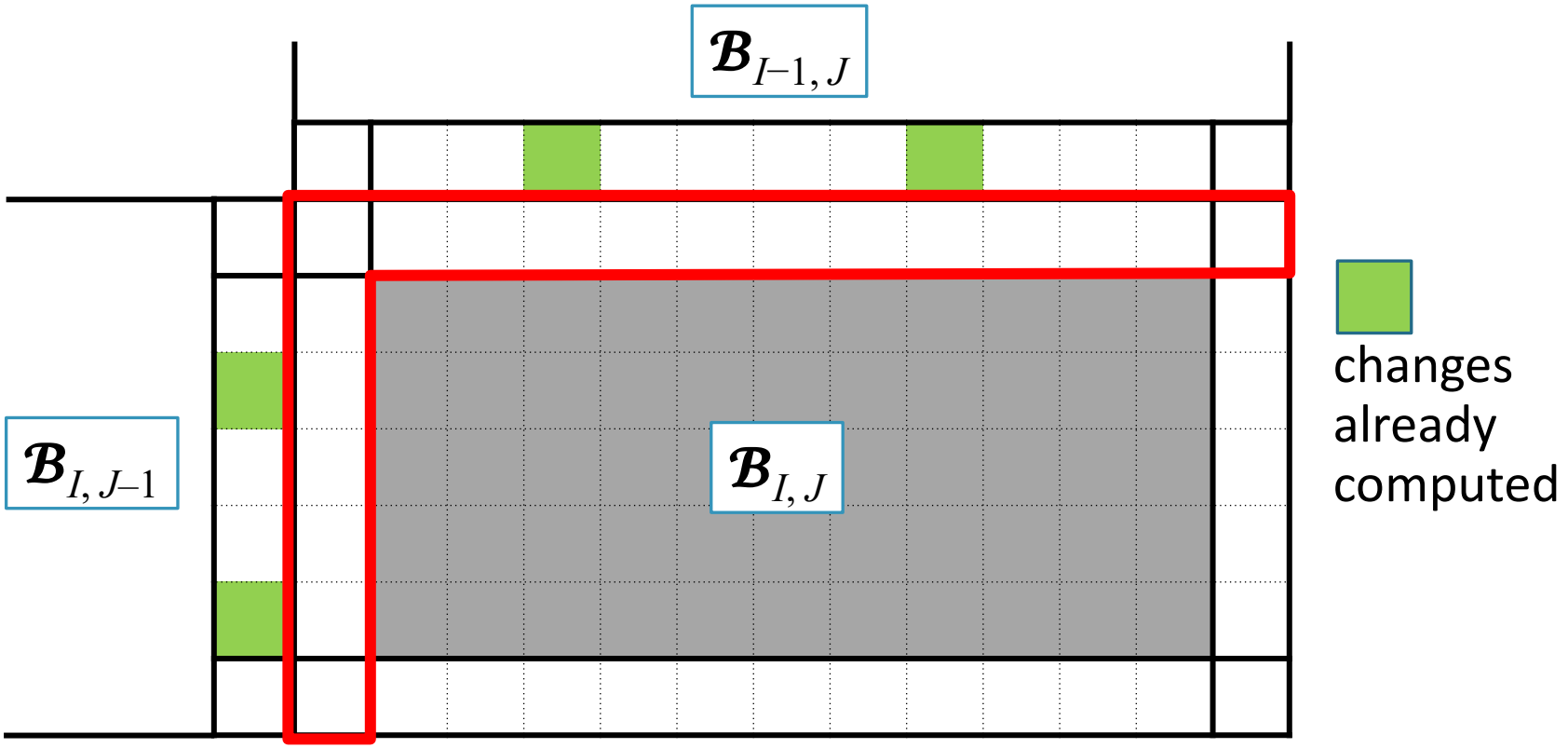
Now we want to detect cells on the boundaries of $\mathcal{B}_{I,J}$ whose values will change in DS'.



Updating Top/Left Boundaries of $\mathcal{B}_{I,J}$

First, consider the top/left boundaries of $\mathcal{B}_{I,J}$.

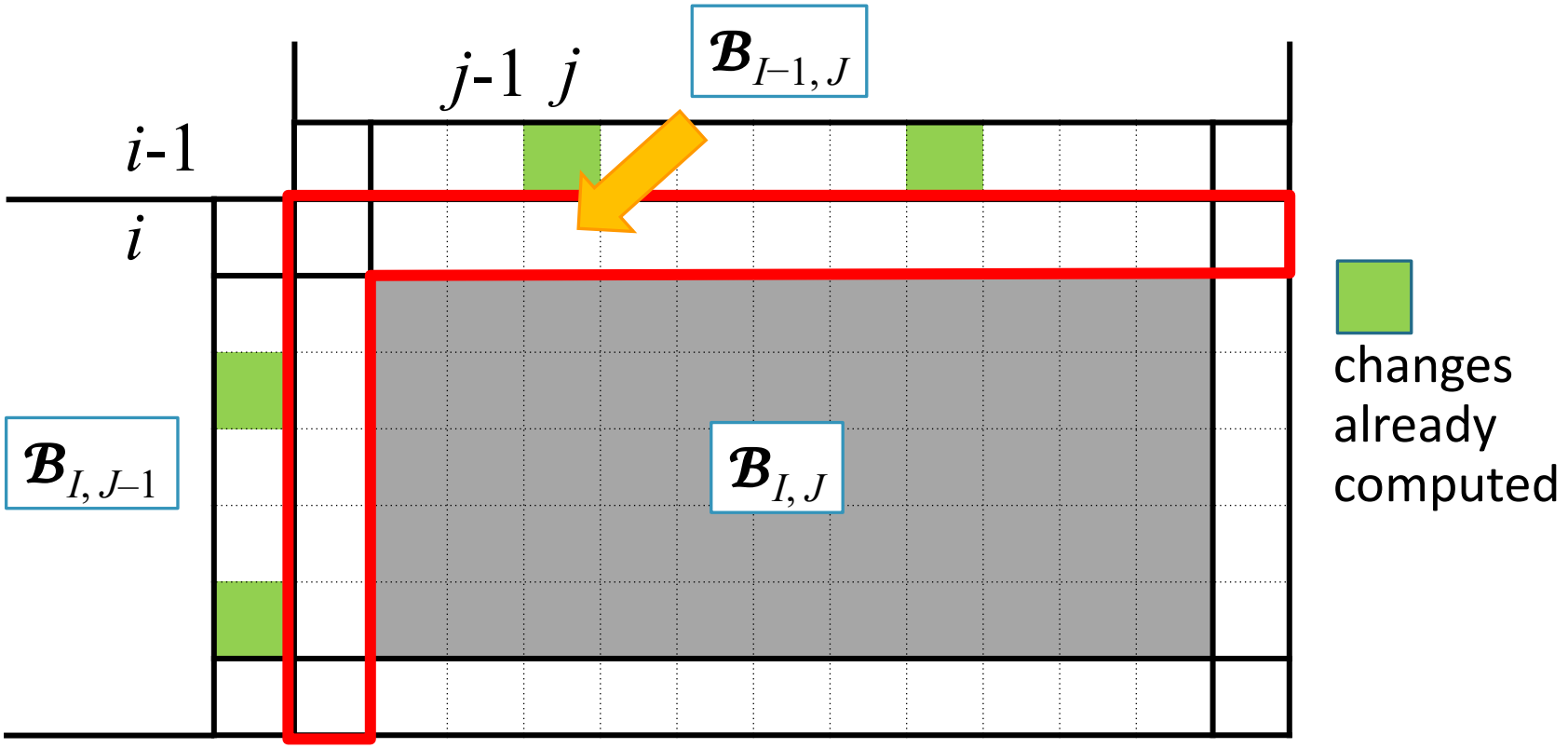
The changes will propagate from the bottom/right boundaries of $\mathcal{B}_{I-1,J}$ and $\mathcal{B}_{I,J-1}$, respectively.



Updating Top/Left Boundaries of $\mathcal{B}_{I,J}$ [cont.]

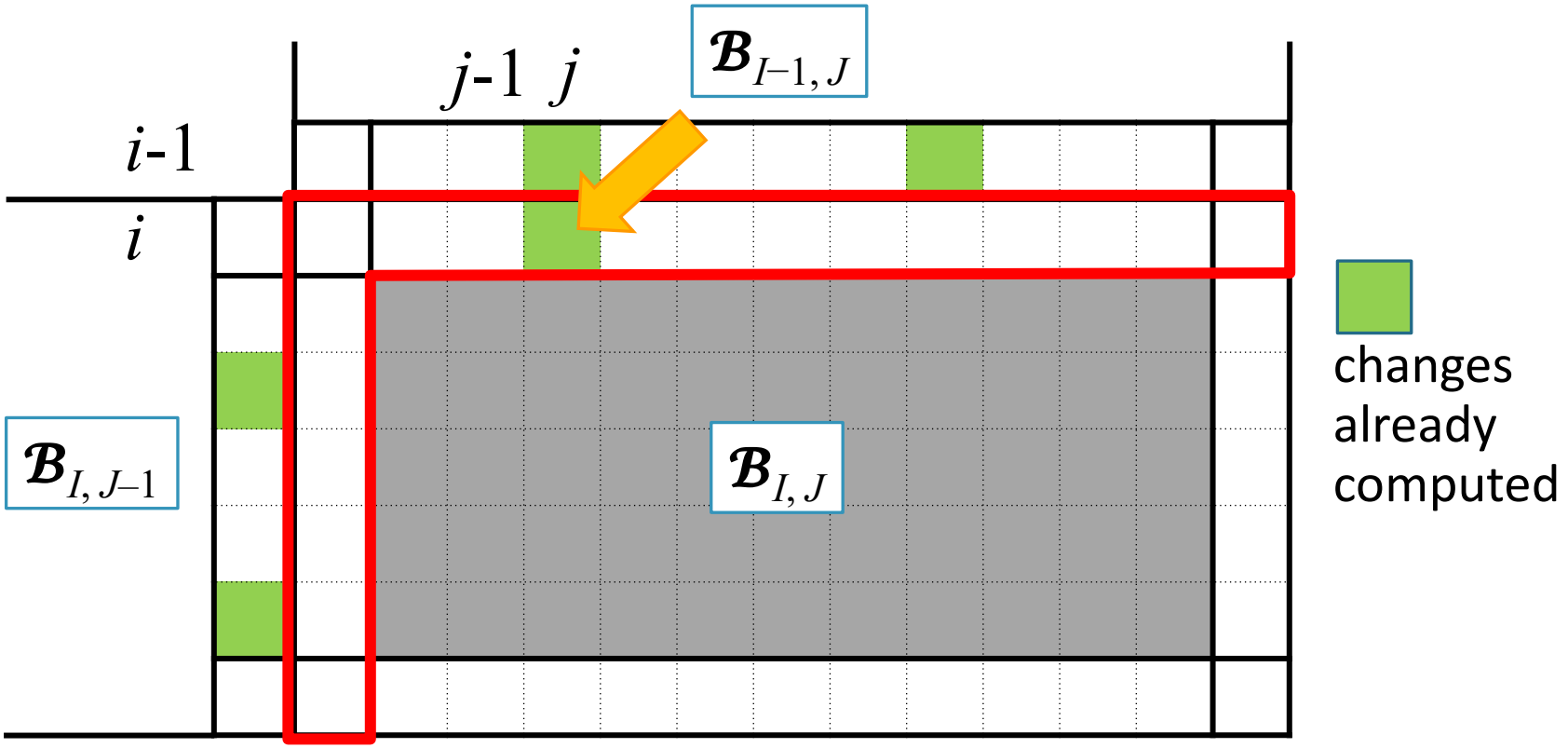
Let (i, j) be a top boundary cell of $\mathcal{B}_{I,J}$ such that the bottom boundary cell $(i\text{-}1, j)$ of $\mathcal{B}_{I-1,J}$ has been changed.

Because we know the values of $DS'[i\text{-}1, j]$ and $DS'[i, j\text{-}1]$, we can compute the value of $DS'[i, j]$ in constant time.



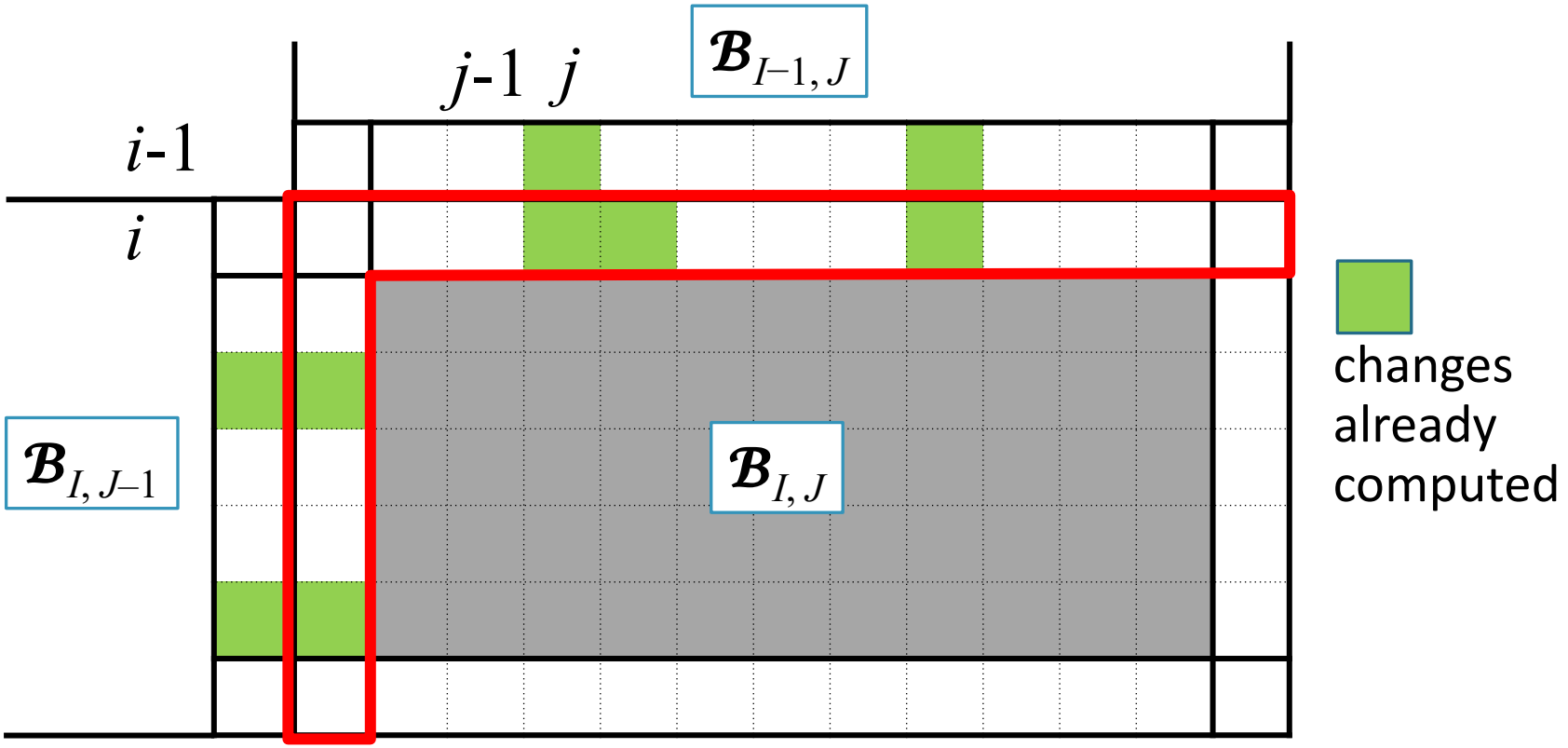
Updating Top/Left Boundaries of $\mathcal{B}_{I,J}$ [cont.]

This way, we can determine whether the change of the upper-neighbor green cell propagates or not in constant time.



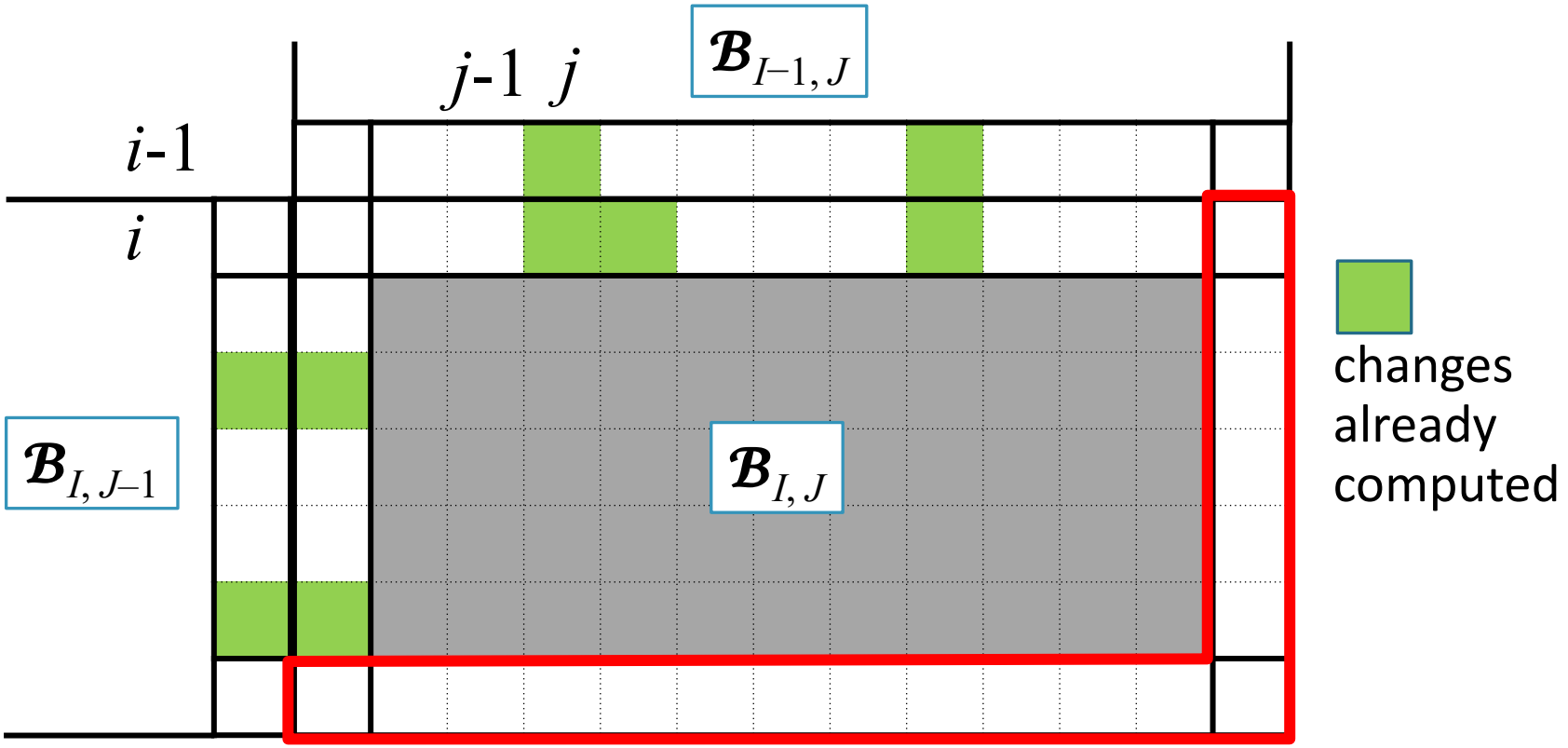
Updating Top/Left Boundaries of $\mathcal{B}_{I,J}$ [cont.]

By repeating this from all the changed cells in the bottom/ right boundaries of $\mathcal{B}_{I-1,J}$ and $\mathcal{B}_{I,J-1}$, we can find all changes in the top/left boundaries of $\mathcal{B}_{I,J}$ in constant time each.



Updating Bottom/Right Boundaries of $\mathcal{B}_{I,J}$

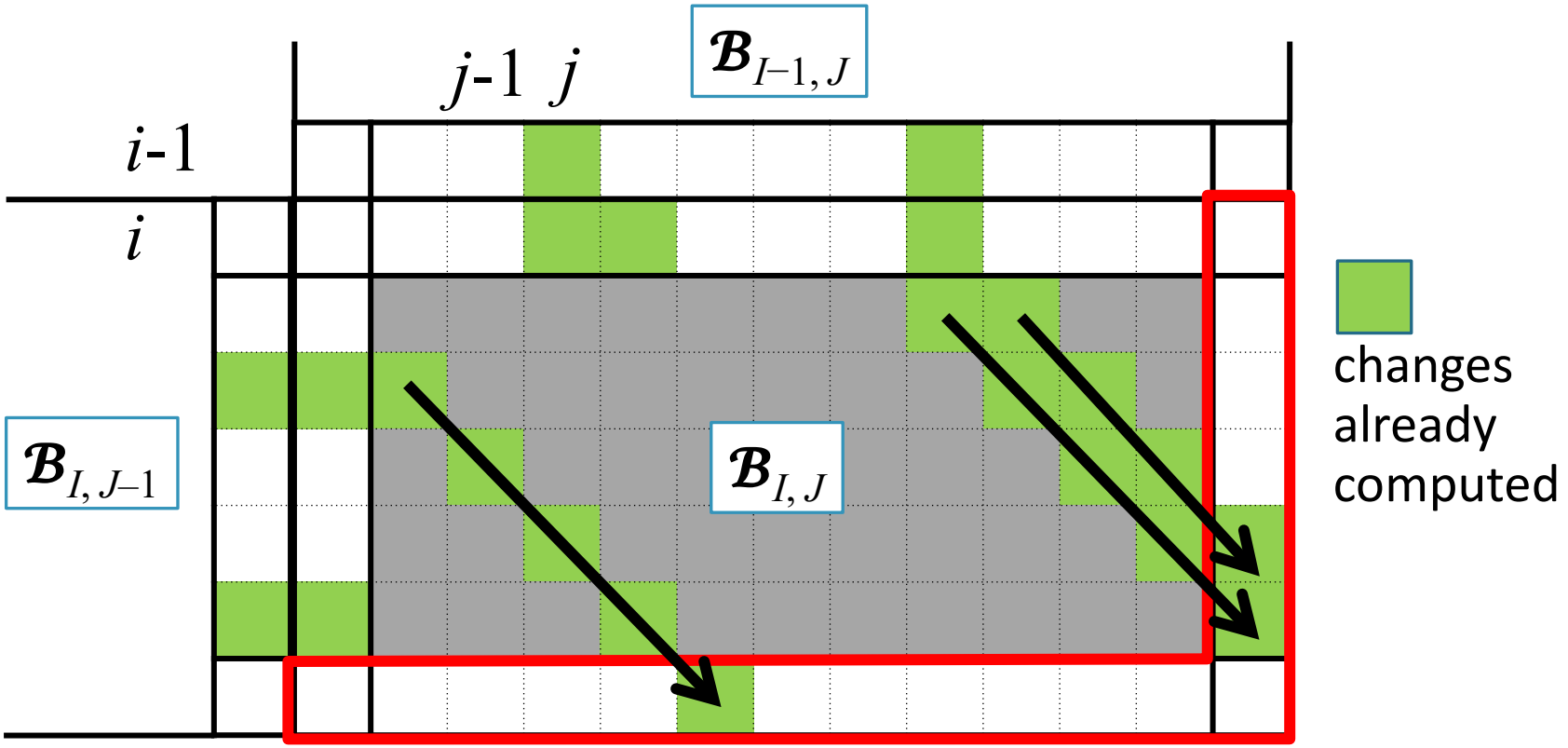
Next, we want to find changed cells on the bottom/right boundaries of $\mathcal{B}_{I,J}$, but we do not want to explicitly process the inside of the box $\mathcal{B}_{I,J}$ (gray zone).



Updating Bottom/Right Boundaries of $\mathcal{B}_{I,J}$ [cont.]

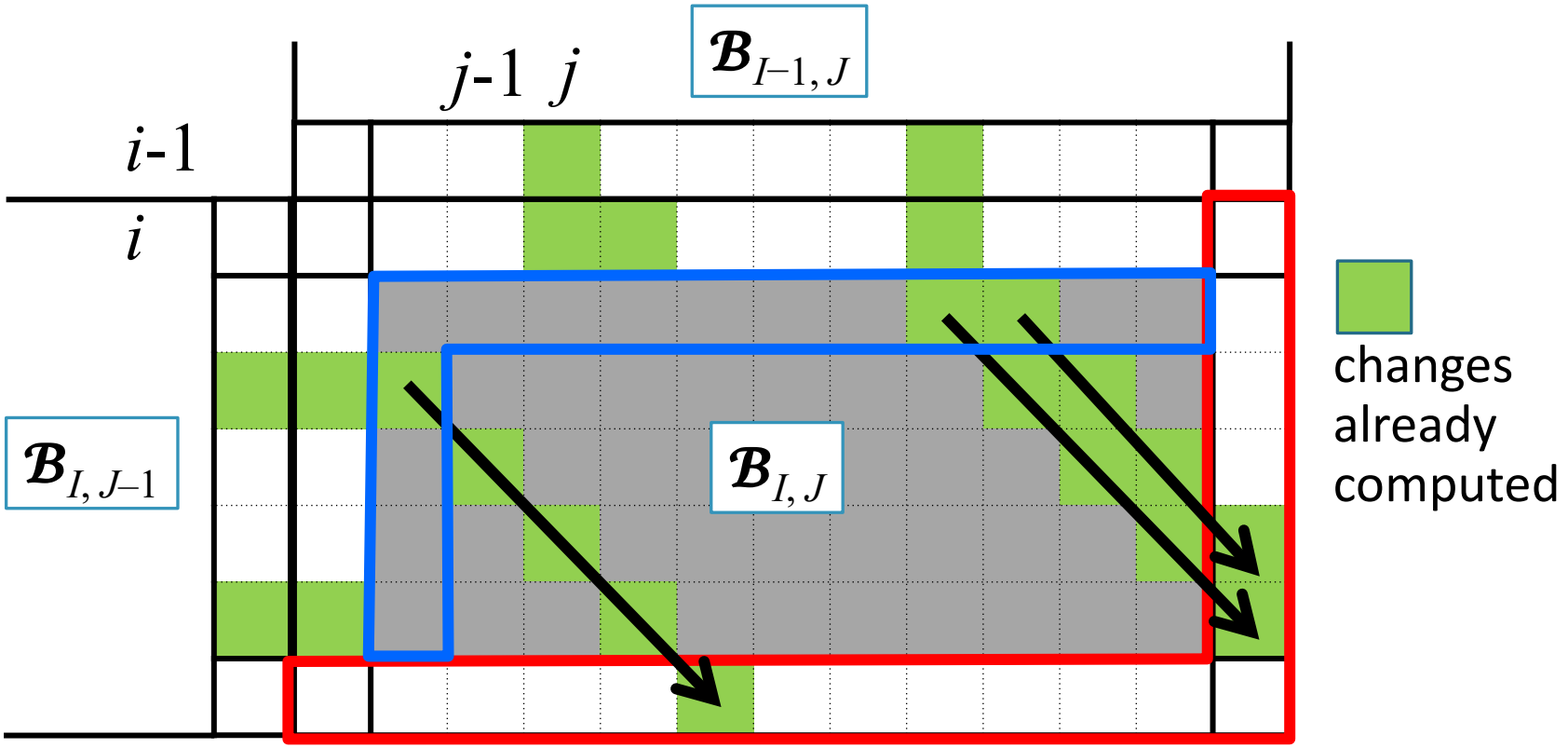
Lemma (Diagonal Propagation)

For any two cells (i, j), $(i+1, j+1)$ in the same box except the top/left boundary, $DR'[i, j] = DR'[i+1, j+1]$.



Updating Bottom/Right Boundaries of $\mathcal{B}_{I,J}$ [cont.]

Due to the diagonal propagation lemma, computing the changed cells in the bottom/right boundaries of $\mathcal{B}_{I,J}$ reduces to computing the changed cells in the top/left-most columns in the gray zone.



Complexity of Our Algorithm

Theorem (D^2TW algorithm)

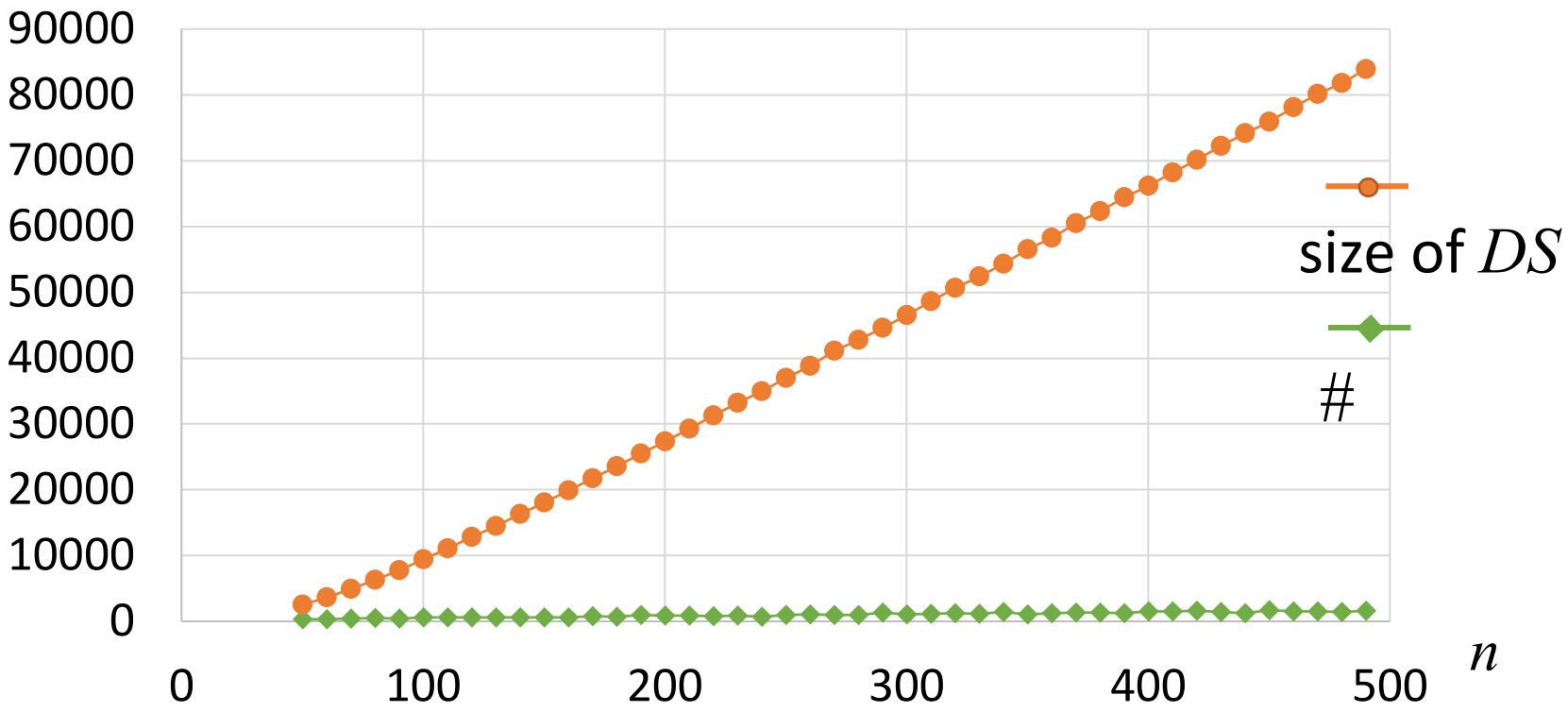
There is a $\Theta(km+ln)$-space data structure for D^2TW which can be updated in $O(m + n + \#)$ time per edit, where $m = |A|$, $n = |B|$, $l = |\mathrm{RLE}(A)|$, $k = |\mathrm{RLE}(B)|$, and $\#$ is the number of cells in DS that change after the edit.

- It is clear that the number of cells that are processed by our algorithm is linear in $\#$.
- Unfortunately, there is a worst-case instance such that $\# = \Omega(km + ln)$.
- Still, $\#$ seems much smaller than $km + ln$ in many cases.

Preliminary Experiments (1)

For randomly generated strings A, B of length $n = 50, \ldots, 500$ with fixed RLE size $k = 50$, we computed

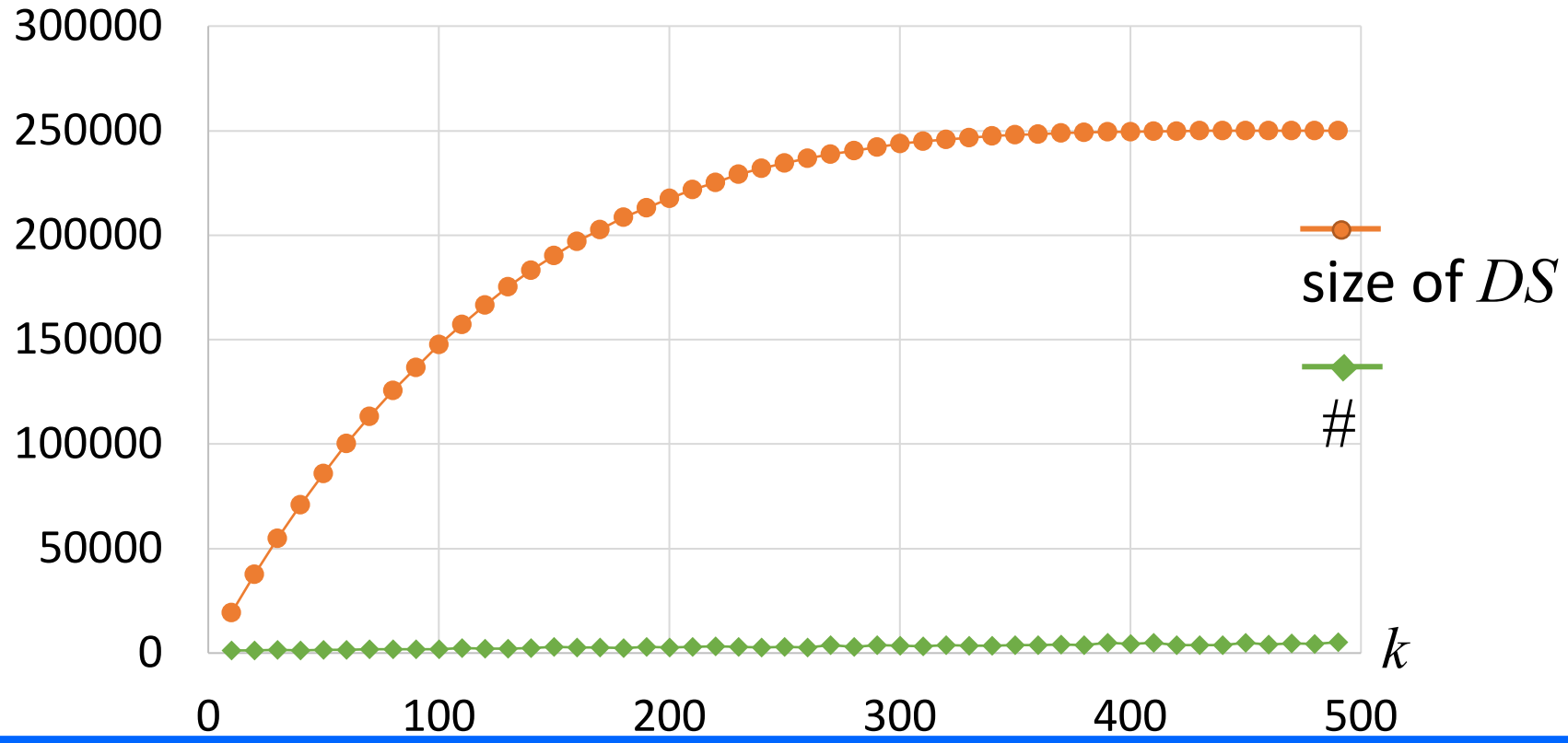
- the size of DS (number of boundary cells)
- the value of $\#$ when $B[1]$ is deleted.



Preliminary Experiments (2)

For randomly generated strings A, B of fixed length $n = 500$ with RLE size $k = 10, \ldots, 500$, we computed

- the size of DS (number of boundary cells)
- the value of $\#$ when $B[1]$ is deleted.



Conclusions and Open Question

We proposed the first DTW algorithm in a dynamic setting, which occupies $\Theta(km + ln)$ space and uses $\mathrm{O}(m + n + \#)$ time per edit operation on the strings.

The value of $\#$ can be as large as $\Theta(km + ln)$ in the worst case, but could be much smaller in practice.

For weighted edit distance with maximum weight c, it is known that $\# = \mathrm{O}(c(m + n))$ [Hyyrö & Inenaga, 2018].

Is there a data structure for DTW which can be updated faster than $\mathrm{O}(km + ln)$ per edit operation?

In our worst-case instance for DTW, $k = c = \max|a_i - b_j|$.