

Sublinear Time Algorithms for Edit Distance

Barna Saha

University of California San Diego



Joint Work with

Elazar Goldenberg

Tomasz Kociumaka

Robert Krauthgamer

Aviad Rubinfeld

Why Sublinear Time?



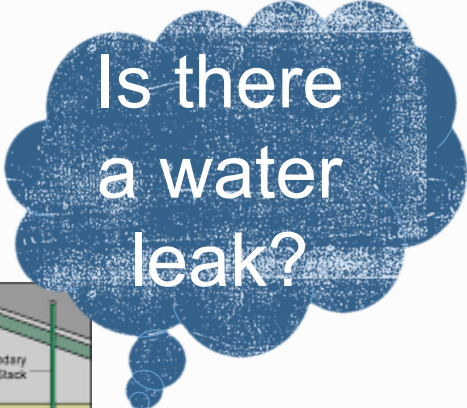
- **Data is massive**
 - Time consuming to access the entire data.
 - Other computational constraints: E.g., space
 - Linear time algorithms are no longer the gold standard



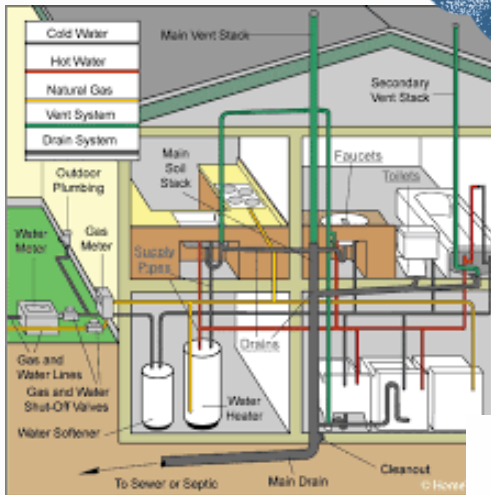
- **Data is complex**
 - Access restriction
 - Policy Constraints, Privacy
- **Only can get a snapshot of data to make inference!**



Only an Approximate View



- Difficult to get exact answer if not impossible when full data is not accessed.
- Often needs randomization as well.
- May need adaptive rounds
- Query complexity vs Running Time



Sublinear Time Algorithms

- **Property Testing:** *Does your data belong to a hypothesis class or very far away from it?*
 - *E.g., “Given a Graph G on n vertices, decide if G is bipartite, or G cannot be made bipartite even after removing an arbitrary subset of at most $\epsilon \binom{n}{2}$ edges.”*
 - Very rich history starting from late nineties
- **Tolerant Testing:** *Approximate distance of a target function to a hypothesis class.*
 - *E.g., “Given a Graph G on n vertices, decide if the number of triangles in G is at most T or above cT where c is the approximation factor.”*
 - Provide trade-offs with approximation factor and query complexity
- **Streaming/Sketching:** Handle data streams. Sublinear space often leads to sublinear time complexity
- **Dynamic Algorithms:** Keeps solution updated all the time. Sublinear update time is a must.



Edit Distance

X: a b b a b a b b a a b a b a b a
Y: a b a b a b a a a a b a b a b

X: a b **b** a b a b **b** a a **b** a b a b a
Y: a b a b a b **a** a a **a** a b a b a **b**

Edit = 4

X: a b **b** a **b** a **b** **b** a a **b** a **b** a **b** a
Y: a b **a** **b** a **b** a a a a **a** **b** a **b** a **b**

Hamming Dist = 12

- EDIT(X,Y): Minimum number of character insertions, deletions or substitutions to convert X to Y.
- Fundamental measure of sequence similarity
- More versatile than Hamming Distance (only substitutions)
- Lot of applications: e.g., in Bioinformatics and Natural Language Processing.

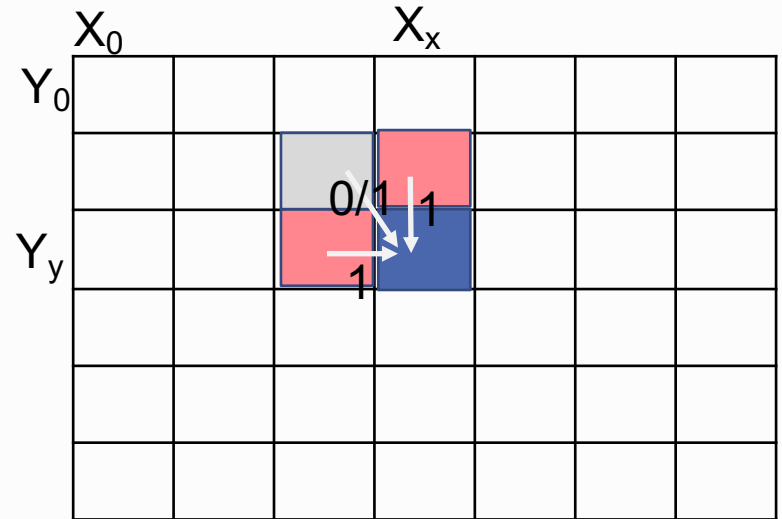


Computing Edit Distance

Exact algorithms

folklore $\mathcal{O}(n^2)$ time

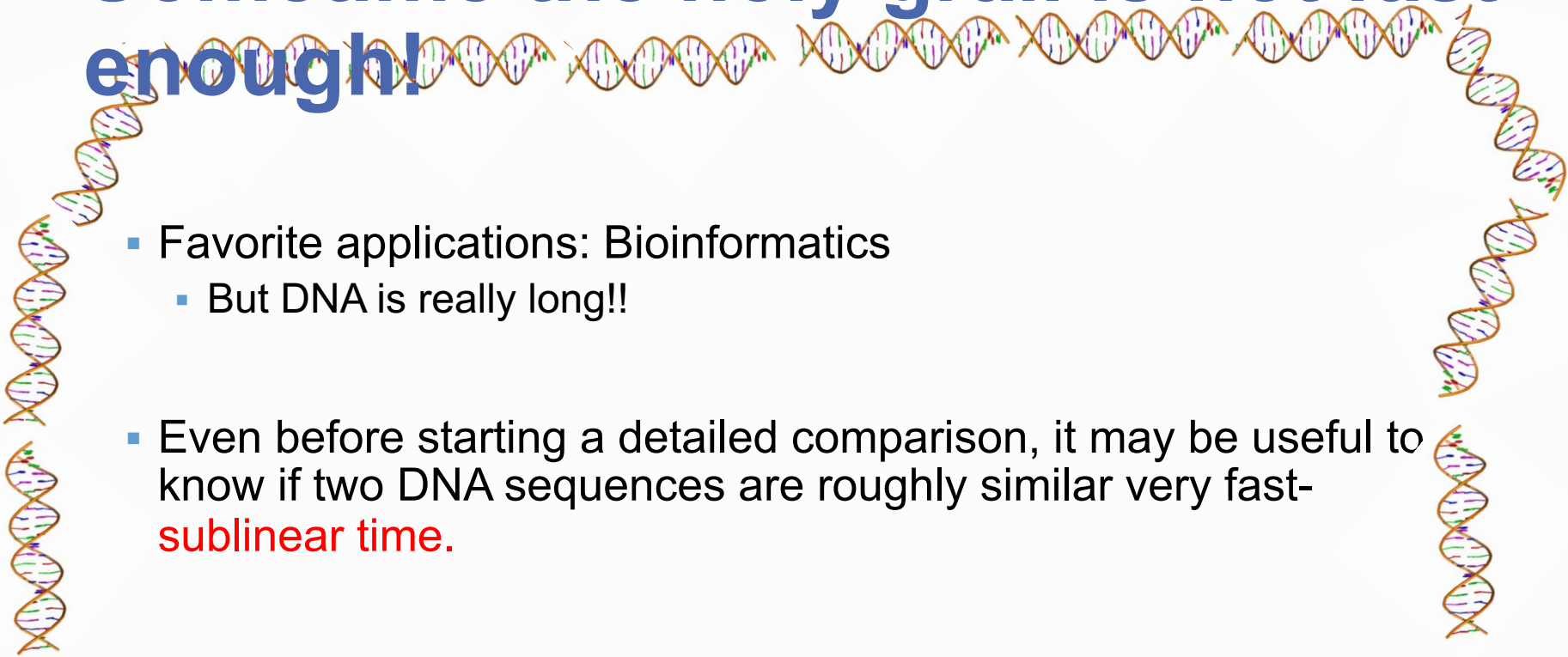
MP80 $\mathcal{O}((n/\log n)^2)$ time



- Truly subquadratic complexity will violate the Strong Exponential Time Hypothesis (SETH)
 - One of the important results in Fine-grained Complexity
- Long line of work on approximation algorithms
 - Typical holy grail: near-linear time algorithm



Sometime the holy grail is not fast enough!



- Favorite applications: Bioinformatics
 - But DNA is really long!!
- Even before starting a detailed comparison, it may be useful to know if two DNA sequences are roughly similar very fast-**sublinear time.**
- Many pairs of DNA sequences may need to be compared- **if we know pair-wise distance is small, we want to compute it exactly in sublinear time.**



Sublinear Time Edit Distance Approximation

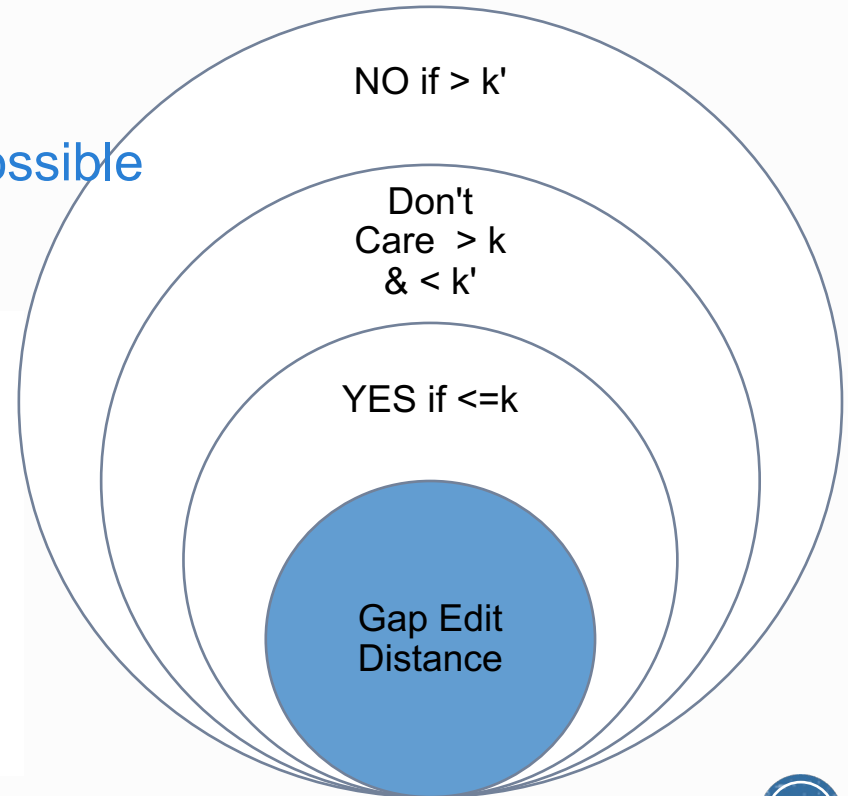
- **Model:**
 - $O(1)$ -time random access to X and Y
 - Monte-Carlo randomization
- **Sublinear time exact algorithm: Impossible**
 - Testing $X=Y$ requires $\Omega(n)$ time!

Gap Edit Distance Problem

Given random access to strings X and Y , and parameters $0 \leq k \leq k' \leq n$, return:

- YES if $ED(X, Y) \leq k$,
- NO if $ED(X, Y) > k'$.

Any answer allowed if $k < ED(X, Y) \leq k'$.



Hamming vs Edit: Sublinear Time

- Hamming Distance
 - Distinguish k vs ck for some constant c
 - Uniform random sample at a rate of $\Theta\left(\frac{\log n}{k}\right)$
- Edit Distance
 - Uniform random sampling cannot distinguish 2 edits from n edits
 - Case 1: **Many Edits**
 - X: a b c d e f g ..
 - Y: h i j k l m o ..
 - EDIT = n
 - Case 2: **Few Edits**
 - X: a b c d e f g ..
 - Y: h a b c d e f ..
 - EDIT = 2



Gap Edit Distance [2003-2020]

Landau & Viskin (JCSS, 1988): $\mathcal{O}(n + k^2)$

- Exact baseline.

Batu, Ergün, Kilian, Magen, Raskhodnikova, Rubinfeld, Sami (STOC 2003): $\mathcal{O}\left(\frac{k^2}{n} + \sqrt{k}\right)$.

- Allows $k' = \Omega(n)$ only.

Andoni, Onak (STOC 2009): $\mathcal{O}\left(\frac{n^{2+o(1)}k}{(k')^2}\right)$.

- Efficient for large k' .

Goldenberg, Krauthgamer, Saha (FOCS 2019): $\tilde{\mathcal{O}}\left(\frac{nk}{k'} + k^3\right)$.

- Efficient for small k .

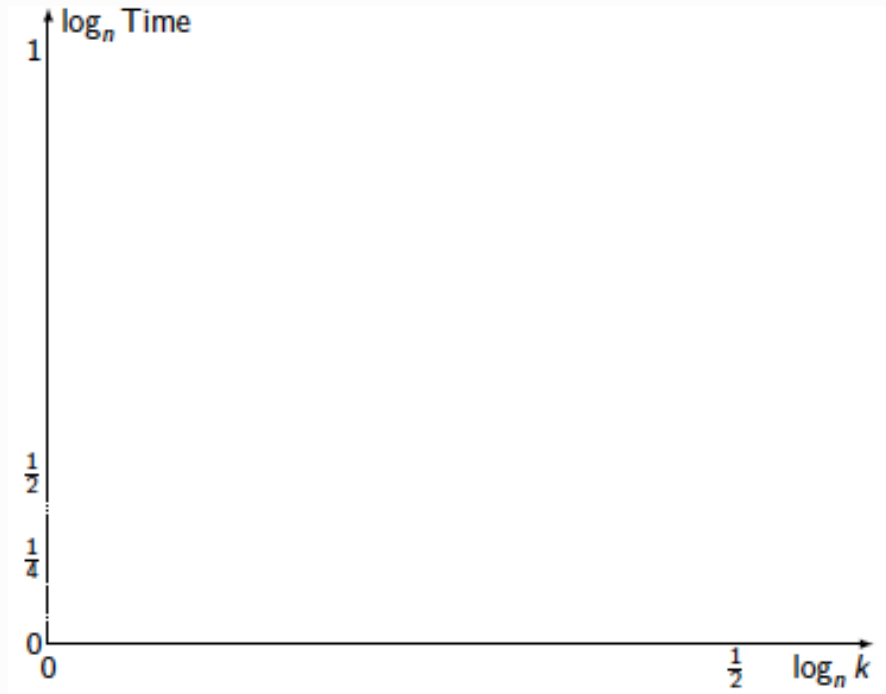
Kociumaka, Saha (FOCS 2020)

$$\tilde{\mathcal{O}}\left(\frac{nk}{k'} + k^2 + \frac{\sqrt{nk^5}}{k'}\right).$$

- Improved running time for medium k and k' .

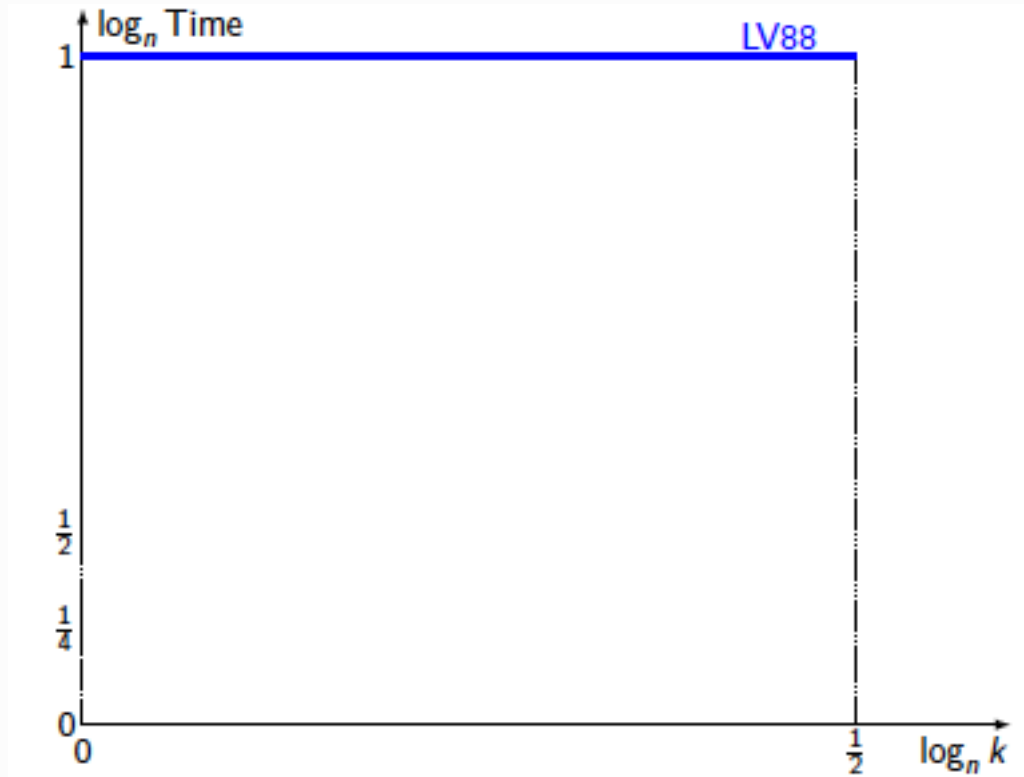


Gap Edit Distance: k vs k^2



Gap Edit Distance: k vs k^2

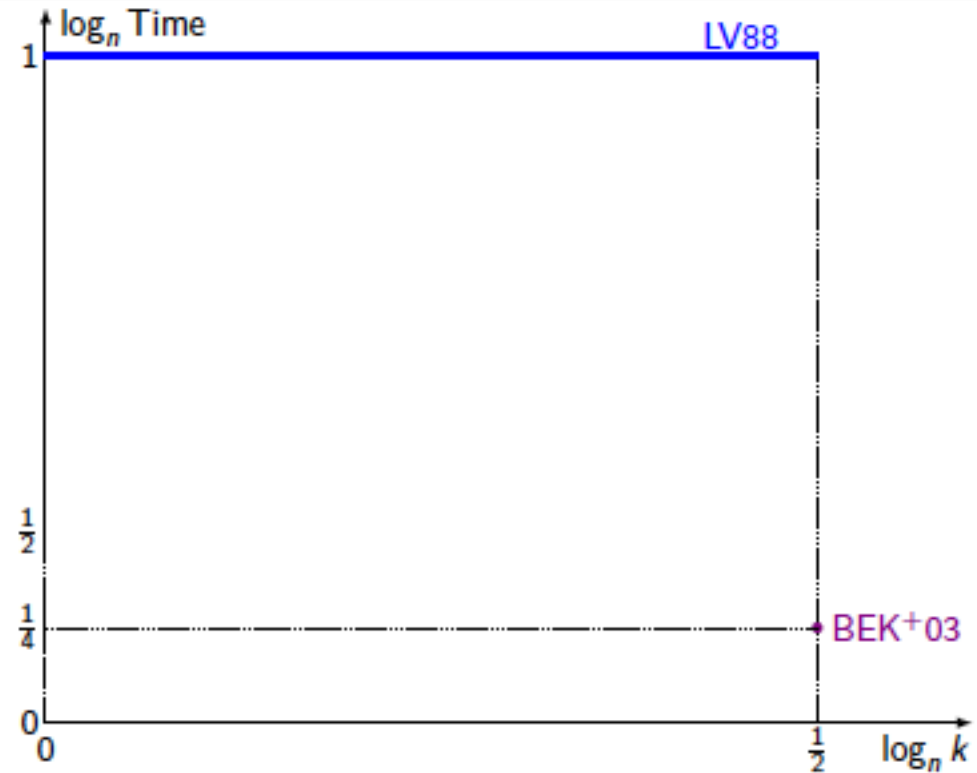
LV88 $\mathcal{O}(n + k^2)$



Gap Edit Distance: k vs k^2

LV88 $\mathcal{O}(n + k^2)$

BEK+03 $\mathcal{O}(n^{1/4})$
for $k = \Theta(n^{1/2})$

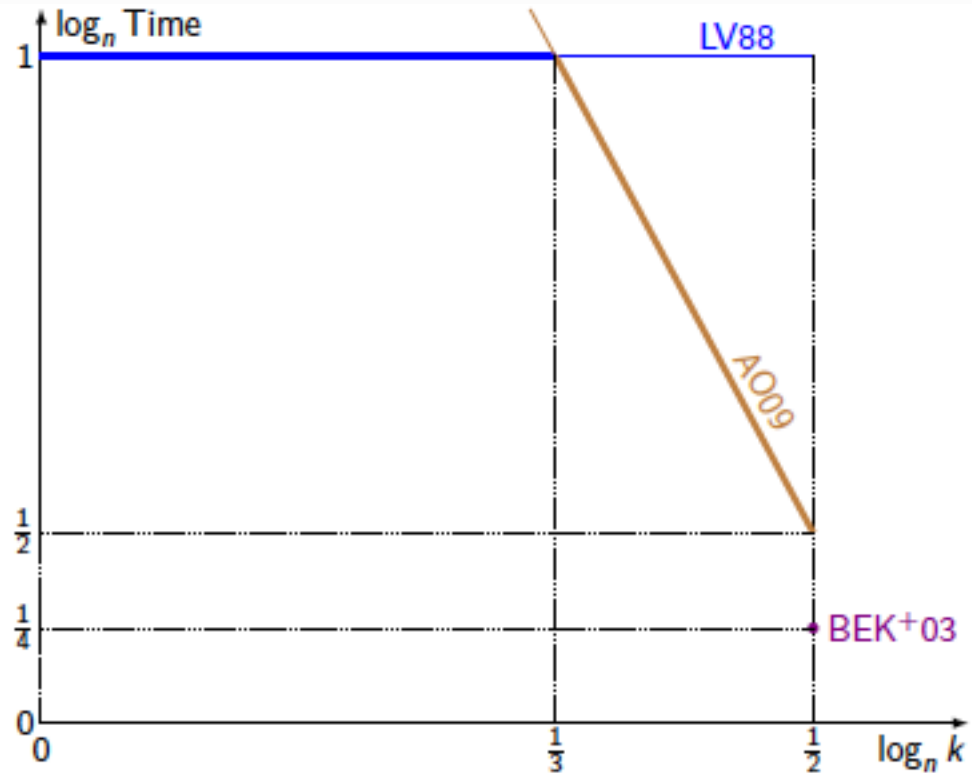


Gap Edit Distance: k vs k^2

LV88 $\mathcal{O}(n + k^2)$

BEK+03 $\mathcal{O}(n^{1/4})$
for $k = \Theta(n^{1/2})$

AO09 $\mathcal{O}\left(\frac{n^{2+o(1)}}{k^3}\right)$



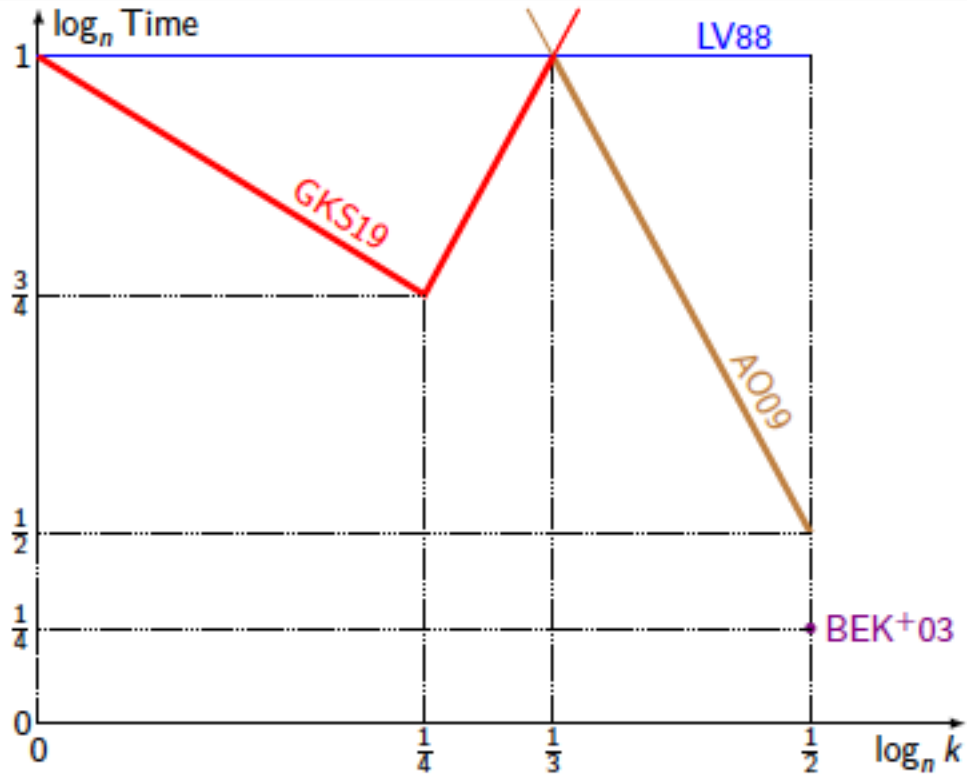
Gap Edit Distance: k vs k^2

LV88 $\mathcal{O}(n + k^2)$

BEK+03 $\mathcal{O}(n^{1/4})$
for $k = \Theta(n^{1/2})$

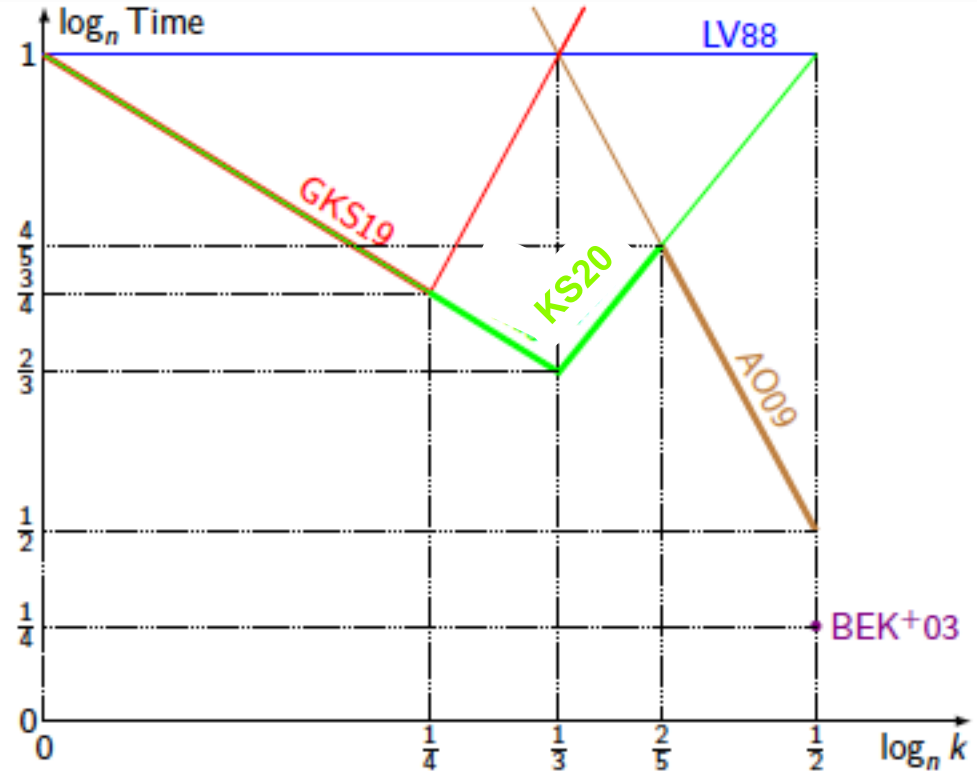
AO09 $\mathcal{O}\left(\frac{n^{2+o(1)}}{k^3}\right)$

GKS19 $\tilde{\mathcal{O}}\left(\frac{n}{k} + k^3\right)$



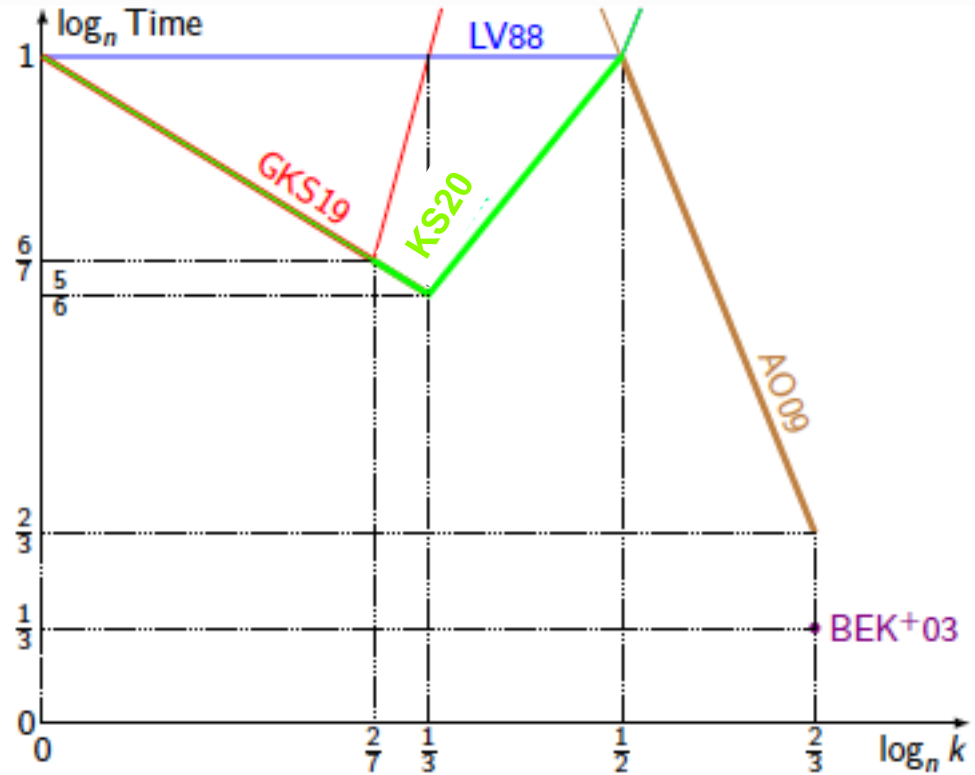
Gap Edit Distance: k vs k^2

- LV88 $\mathcal{O}(n + k^2)$
- BEK+03 $\mathcal{O}(n^{1/4})$
for $k = \Theta(n^{1/2})$
- AO09 $\mathcal{O}\left(\frac{n^{2+o(1)}}{k^3}\right)$
- GKS19 $\tilde{\mathcal{O}}\left(\frac{n}{k} + k^3\right)$
- KS20 $\tilde{\mathcal{O}}\left(\frac{n}{k} + k^2\right)$



Gap Edit Distance: k vs $k^{1.5}$

- LV88 $\mathcal{O}(n + k^2)$
- BEK⁺03 $\mathcal{O}(n^{1/3})$
for $k = \Theta(n^{2/3})$
- AO09 $\mathcal{O}\left(\frac{n^{2+o(1)}}{k^2}\right)$
- GKS19 $\tilde{\mathcal{O}}\left(\frac{n}{\sqrt{k}} + k^3\right)$
- KS20 $\tilde{\mathcal{O}}\left(\frac{n}{\sqrt{k}} + k^2 + k\sqrt{n}\right)$



Gap Edit Distance [2021-2022]

- Can we remove the polynomial dependency on k ?
 - Goldenberg, Kociumaka, Krauthgamer, Saha, FOCS'22
 - k vs k^c : $\tilde{O}(n/k^{c-0.5})$
 - Optimal for nonadaptive algorithms
- Can we match the Hamming distance bound of $\frac{n}{kc}$ for k vs k^c ?
 - Bringmann, Cassis, Fischer, Nakos, STOC'22
 - k vs k^c : $\tilde{O}(n/k^c + n^{0.8} + k^4)$
 - Achieves Hamming distance bound for $k \leq n^{0.1}$ for k vs k^2
 - Can solve subpolynomial gap problem in sublinear regime for sufficiently small k



Gap Edit Distance: k vs k^2

• [BEK+03]: $\tilde{O}(\sqrt{k})$ for $k = \Theta(\sqrt{n})$

----- [AO12]: $\hat{O}(\frac{n^2}{k^3})$

----- [GKS19]: $\tilde{O}(\frac{n}{k} + k^3)$

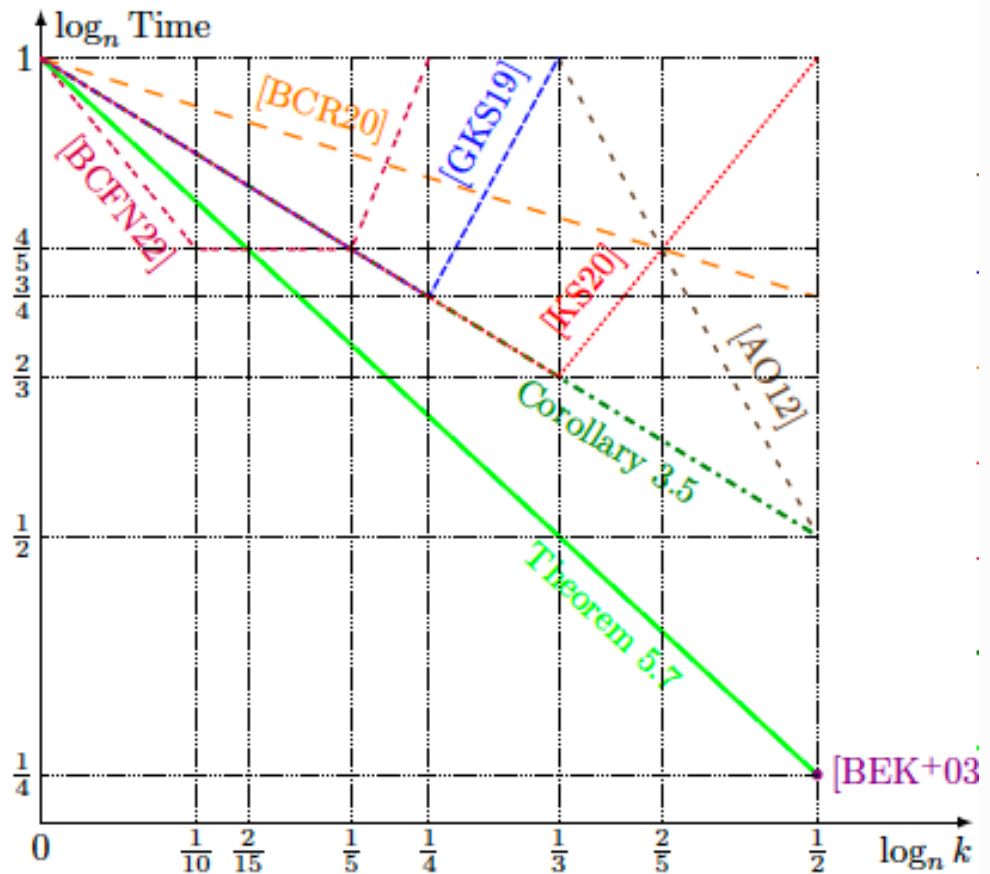
----- [BCR20]: $\mathcal{O}(\frac{n}{\sqrt{k}})$

----- [KS20]: $\tilde{O}(\frac{n}{k} + k^2)$

----- [BCFN22]: $\hat{O}(\frac{n}{k^2} + n^{0.8} + k^4)$

----- Corollary 3.5: $\hat{O}(\frac{n}{k})$

----- Theorem 5.7: $\tilde{O}(\frac{n}{k\sqrt{k}})$



GAP EDIT DISTANCE: k VS k^2
IN $\tilde{O}\left(\frac{n}{k} + k^2\right)$ TIME

Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

$D[x][y] = ED(X[0..x], Y[0..y]).$

	a	b	b	a	b	a	b	b	a	a	b	b	b	a	a	b
a																
b																
a																
b																
a																
b																
a																
a																
a																
a																
b																
b																
b																
a																
a																
b																



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}$.

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

	a	b	b	a	b	a	b	b	a	a	b	b	b	a	a	b	
a																	
b																	
a																	
b																	
a																	
b																	
a																	
a																	
a																	
a																	
b																	
b																	
b																	
a																	
a																	
b																	



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}$.

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

	a	b	b	a	b	a	b	b	a	a	b	b	b	a	a	b
0																
a																
b																
a																
b																
a																
b																
a																
a																
a																
a																
b																
b																
b																
a																
a																
b																



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

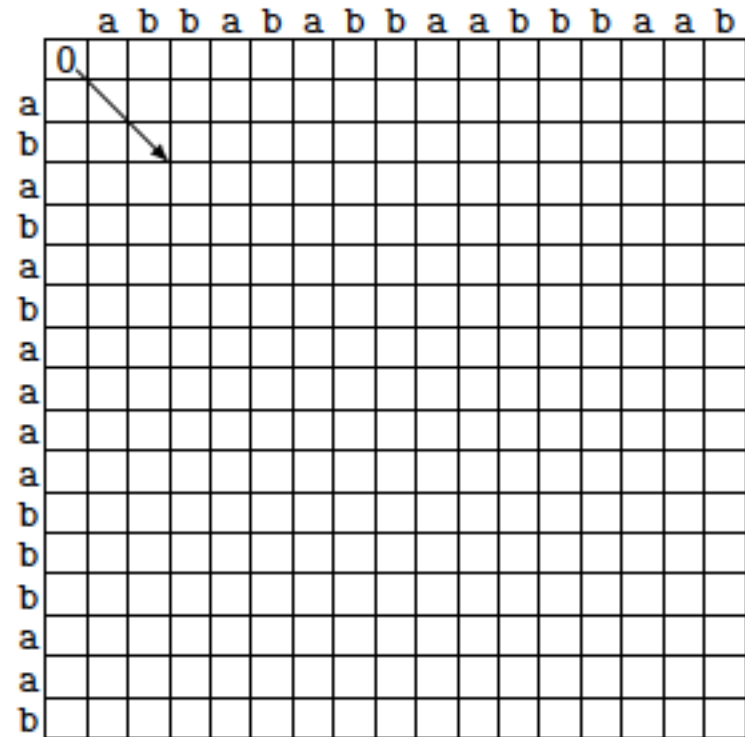
DP table:

$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}$.

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}$.

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

	a	b	b	a	b	a	b	b	a	a	b	b	b	a	a	b	
0																	
a																	
b																	
a																	
b																	
a																	
b																	
a																	
a																	
a																	
a																	
b																	
b																	
b																	
a																	
a																	
b																	



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

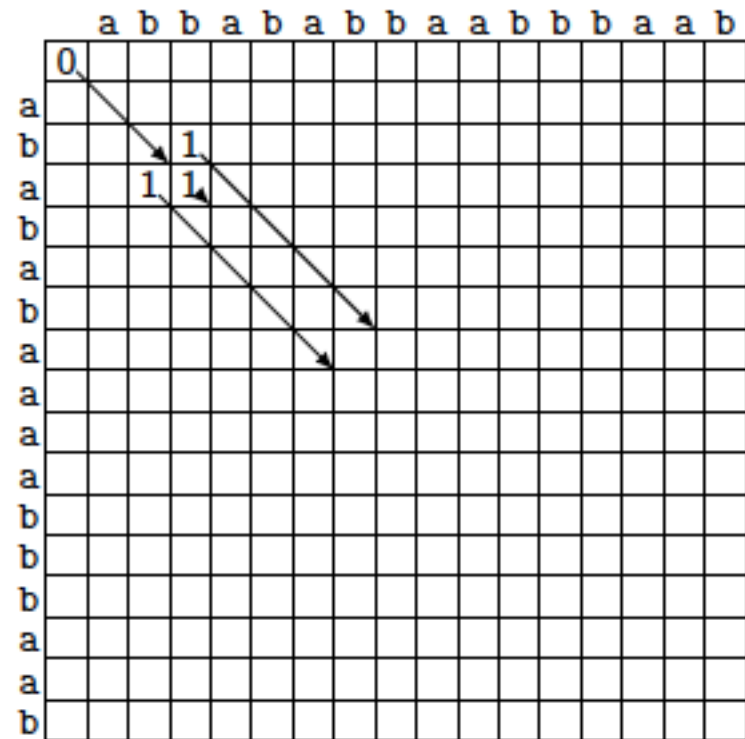
$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}$.

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

- 1 Identify the furthest cell with an $(i - 1)$ -valued neighbor.
- 2 Proceed forward using an LCE query.



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

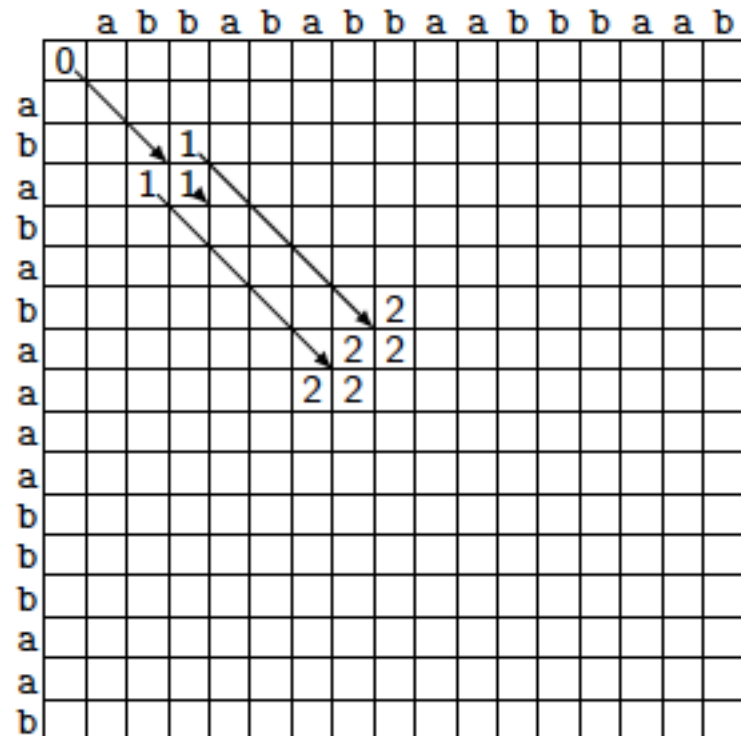
$D[x][y] = ED(X[0..x], Y[0..y]).$

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}.$

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

- 1 Identify the furthest cell with an $(i - 1)$ -valued neighbor.
- 2 Proceed forward using an LCE query.



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

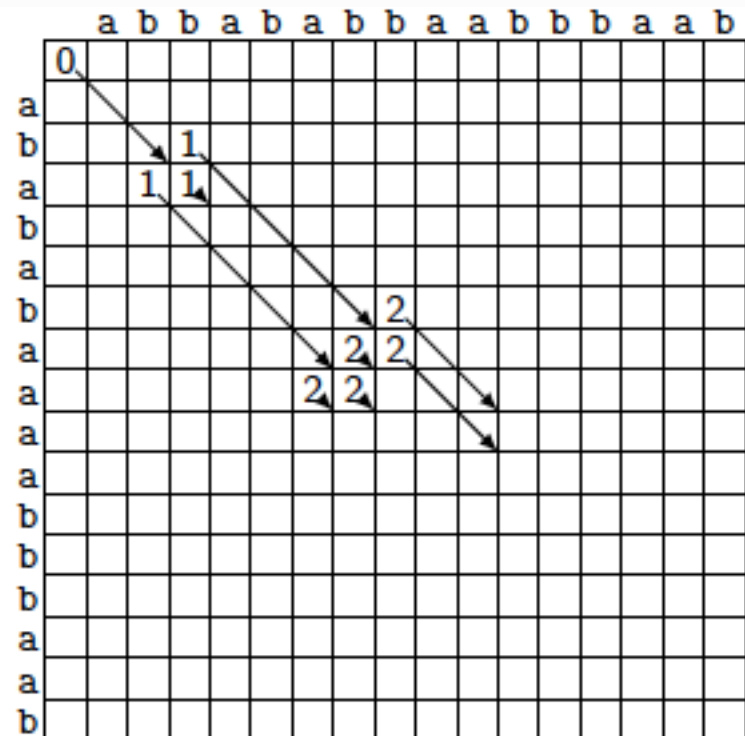
$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}$.

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

- 1 Identify the furthest cell with an $(i - 1)$ -valued neighbor.
- 2 Proceed forward using an LCE query.



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

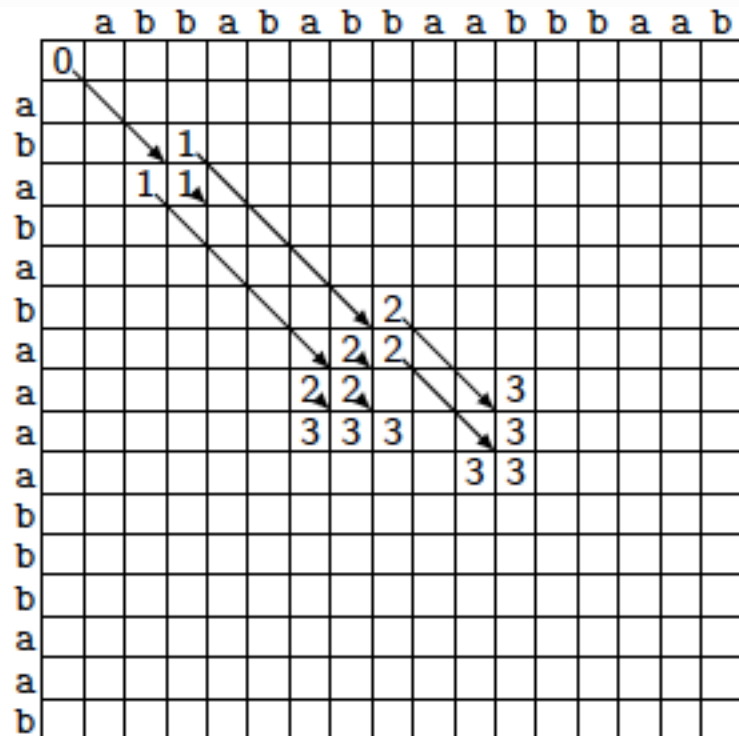
$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}$.

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

- 1 Identify the furthest cell with an $(i - 1)$ -valued neighbor.
- 2 Proceed forward using an LCE query.



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

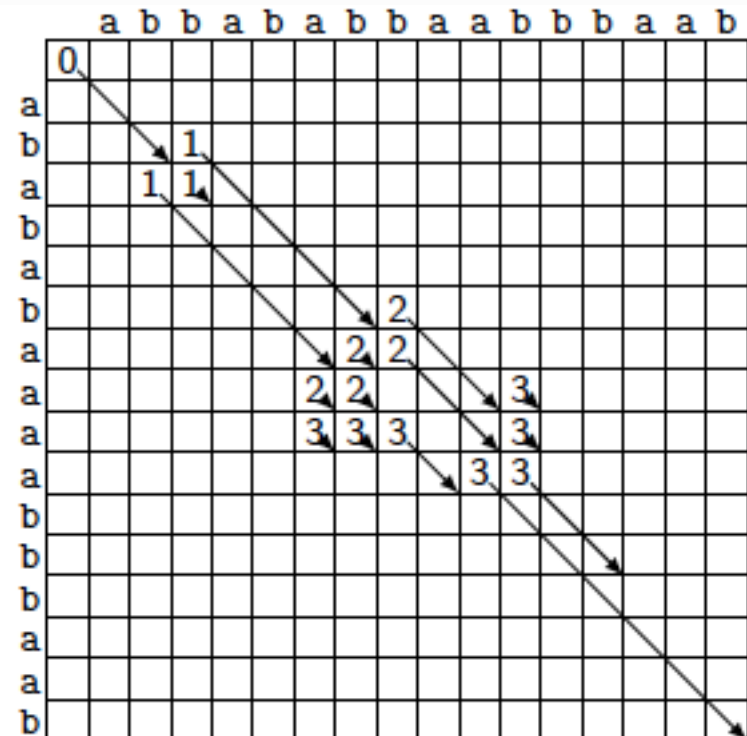
$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}$.

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

- 1 Identify the furthest cell with an $(i - 1)$ -valued neighbor.
- 2 Proceed forward using an LCE query.



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}$.

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

- 1 Identify the furthest cell with an $(i - 1)$ -valued neighbor.
- 2 Proceed forward using an LCE query.



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $\mathcal{O}(n + k^2)$ time.

DP table:

$D[x][y] = ED(X[0..x], Y[0..y]).$

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}.$

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

- 1 Identify the furthest cell with an $(i - 1)$ -valued neighbor.
- 2 Proceed forward using an LCE query.

Longest Common Extension queries:

$\mathcal{O}(1)$ time after $\mathcal{O}(n)$ -time preprocessing.



Algorithm of Goldenberg, Krauthgamer & Saha, FOCS 2019

Main idea:

Use the Landau–Vishkin algorithm, allow $< k$ missed mismatches per LCE query.

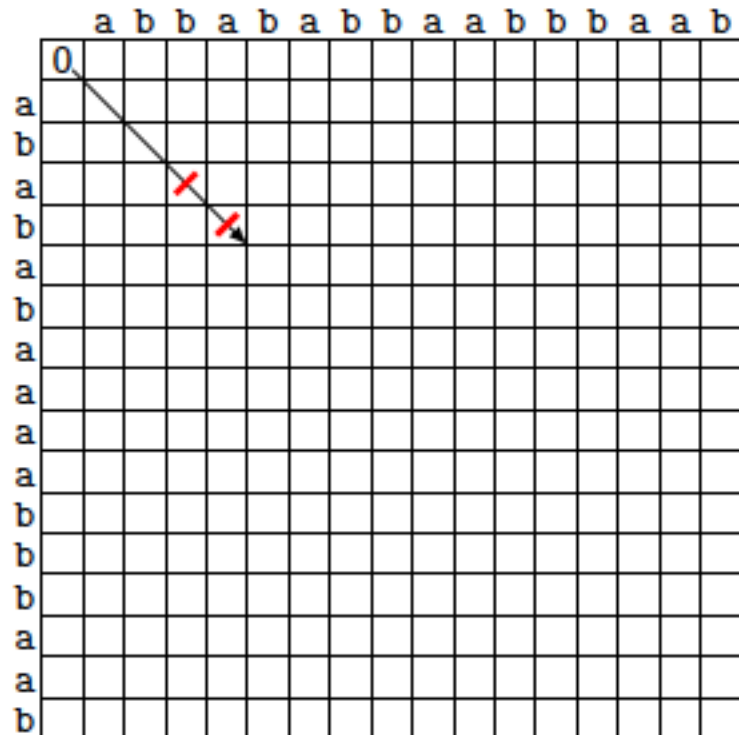
	a	b	b	a	b	a	b	b	a	a	b	b	b	a	a	b
0																
a																
b																
a																
b																
a																
b																
a																
a																
a																
a																
b																
b																
b																
a																
a																
b																



Algorithm of Goldenberg, Krauthgamer & Saha, FOCS 2019

Main idea:

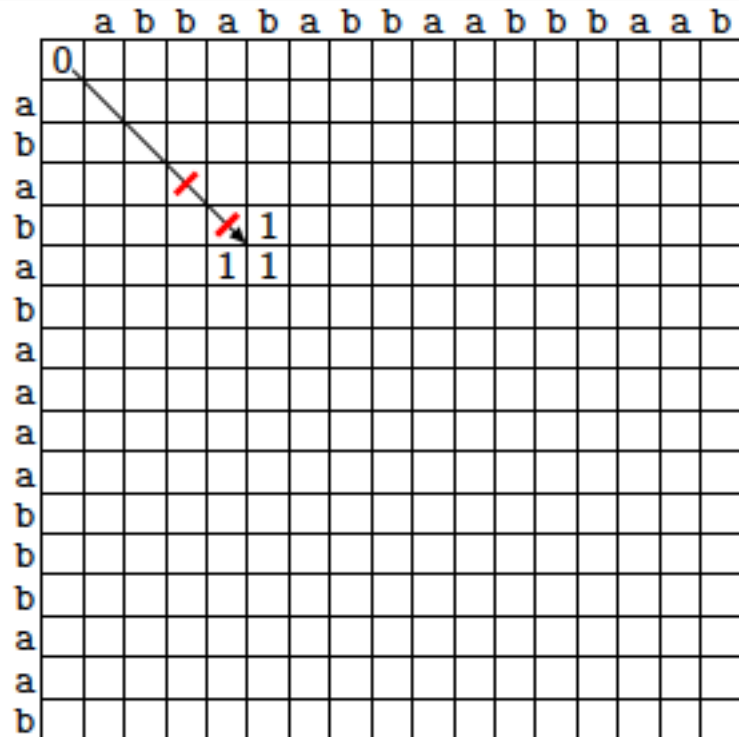
Use the Landau–Vishkin algorithm, allow $< k$ missed mismatches per LCE query.



Algorithm of Goldenberg, Krauthgamer & Saha, FOCS 2019

Main idea:

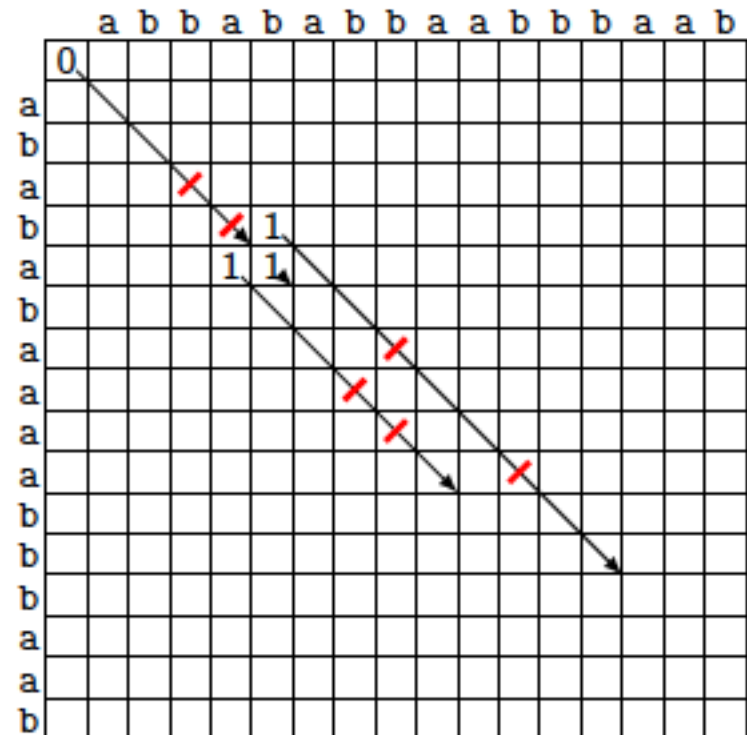
Use the Landau–Vishkin algorithm, allow $< k$ missed mismatches per LCE query.



Algorithm of Goldenberg, Krauthgamer & Saha, FOCS 2019

Main idea:

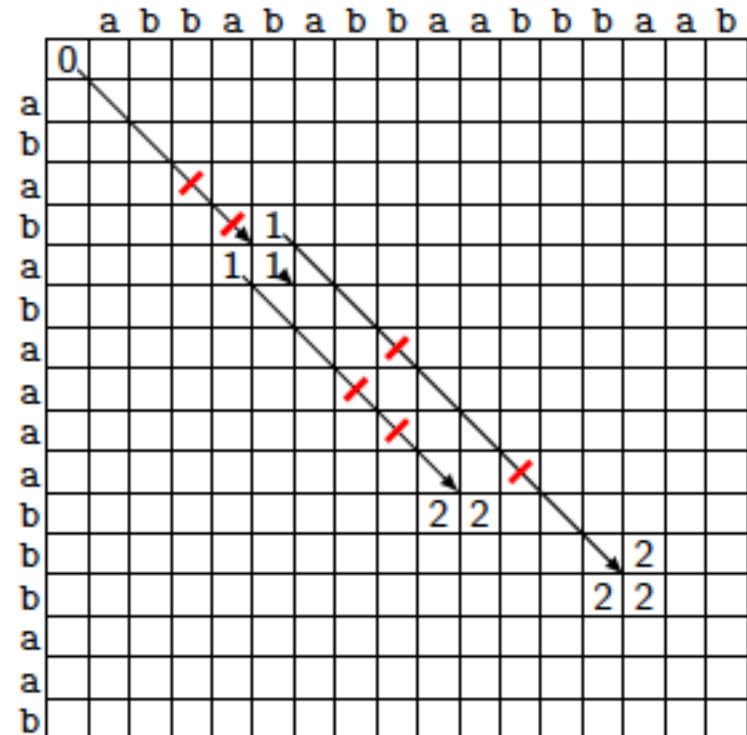
Use the Landau–Vishkin algorithm, allow $< k$ missed mismatches per LCE query.



Algorithm of Goldenberg, Krauthgamer & Saha, FOCS 2019

Main idea:

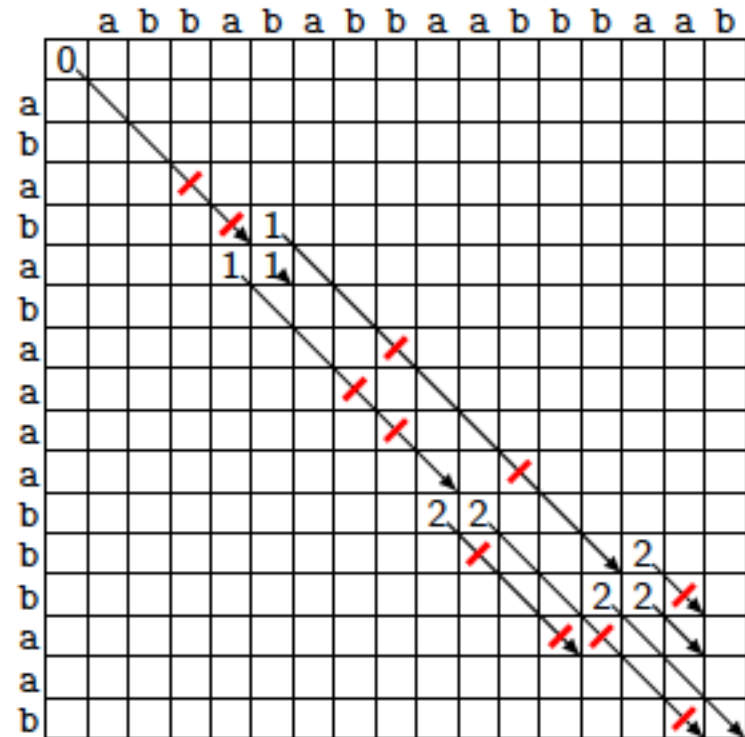
Use the Landau–Vishkin algorithm, allow $< k$ missed mismatches per LCE query.



Algorithm of Goldenberg, Krauthgamer & Saha, FOCS 2019

Main idea:

Use the Landau–Vishkin algorithm, allow $< k$ missed mismatches per LCE query.



Algorithm of Goldenberg, Krauthgamer & Saha, FOCS 2019

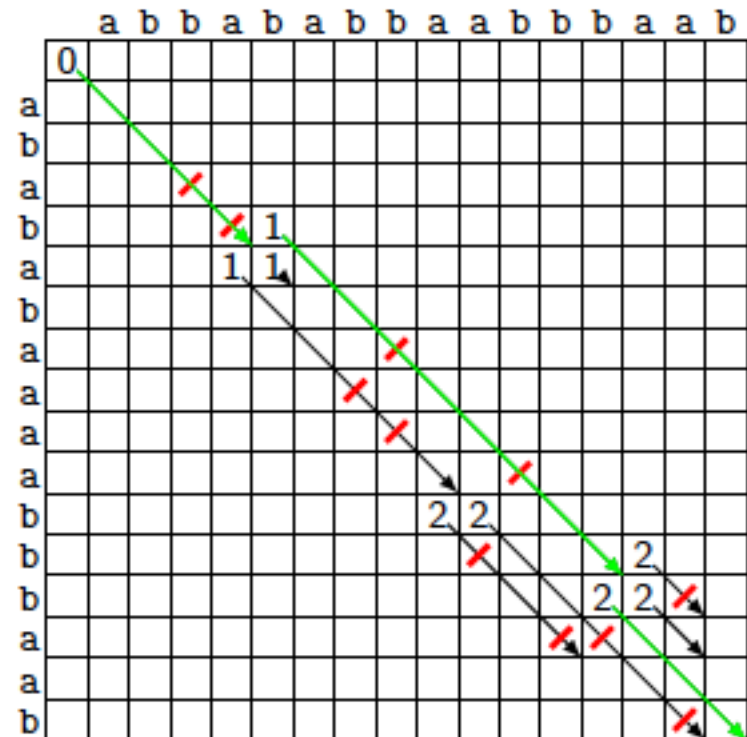
Main idea:

Use the Landau–Vishkin algorithm, allow $< k$ missed mismatches per LCE query.

Guarantees:

$ED(X, Y) \leq k \Rightarrow \text{YES}$

$\text{YES} \Rightarrow ED(X, Y) = \mathcal{O}(k^2)$



Algorithm of Goldenberg, Krauthgamer & Saha, FOCS 2019

Main idea:

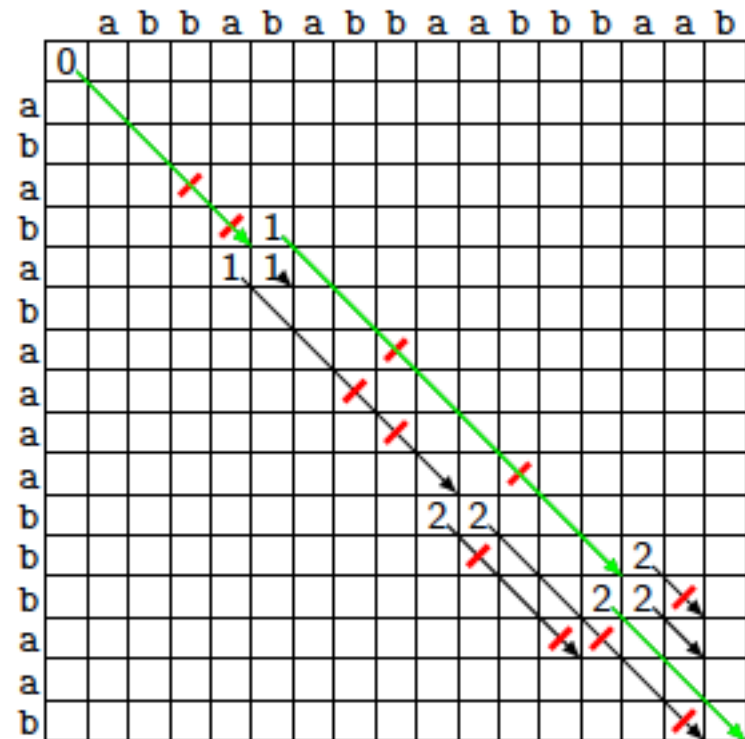
Use the Landau–Vishkin algorithm, allow $< k$ missed mismatches per LCE query.

Guarantees:

$ED(X, Y) \leq k \Rightarrow \text{YES}$

$\text{YES} \Rightarrow ED(X, Y) = \mathcal{O}(k^2)$

Naive implementation: $\tilde{O}(n + k^2)$ time.



Algorithm of Goldenberg, Krauthgamer & Saha, FOCS 2019

Main idea:

Use the Landau–Vishkin algorithm, allow $< k$ missed mismatches per LCE query.

Guarantees:

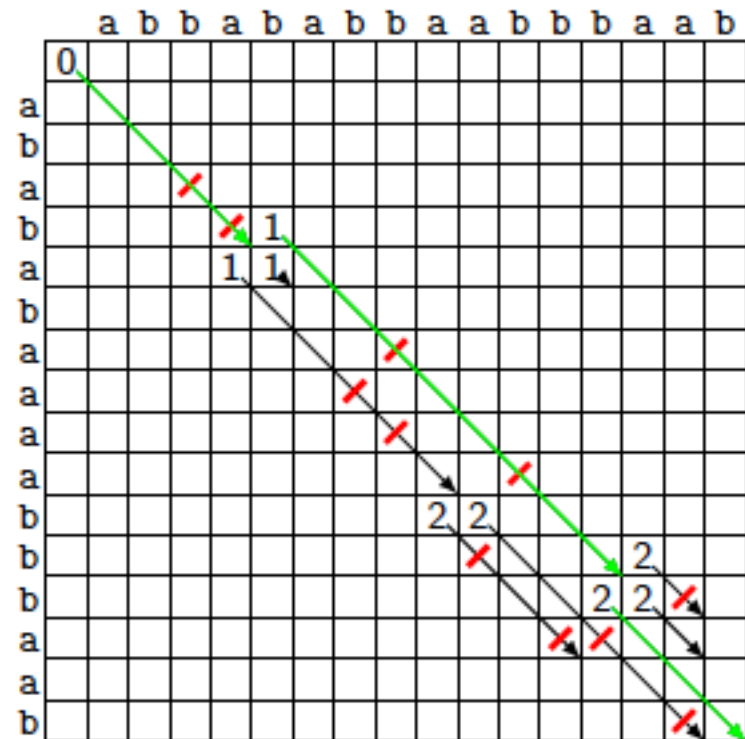
$ED(X, Y) \leq k \Rightarrow \text{YES}$

$\text{YES} \Rightarrow ED(X, Y) = O(k^2)$

Naive implementation: $\tilde{O}(n + k^2)$ time.

May need to look at nk cells in the DP table.

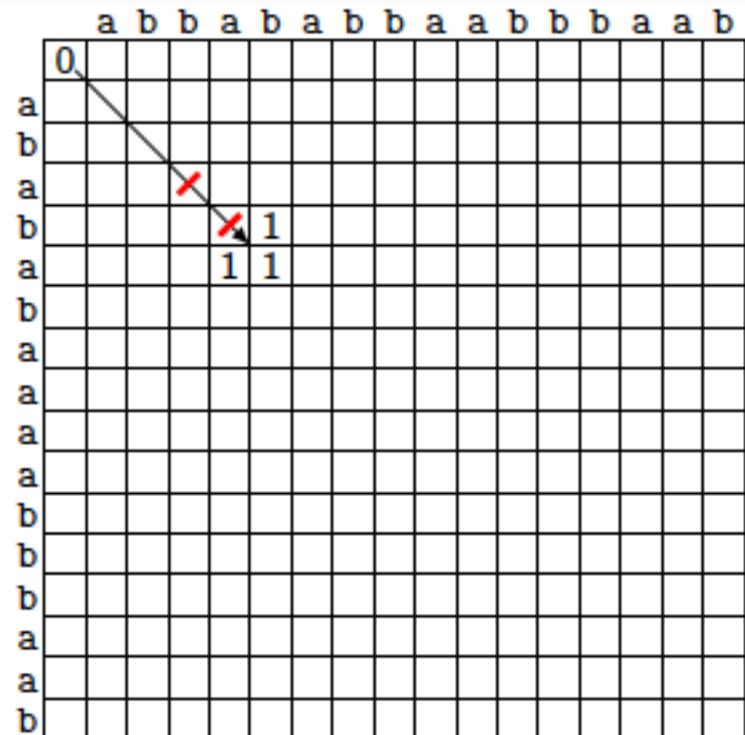
Sample at a rate of $1/k \sim O(n)$



Algorithm of Kociumaka & Saha, FOCS'20

Main idea:

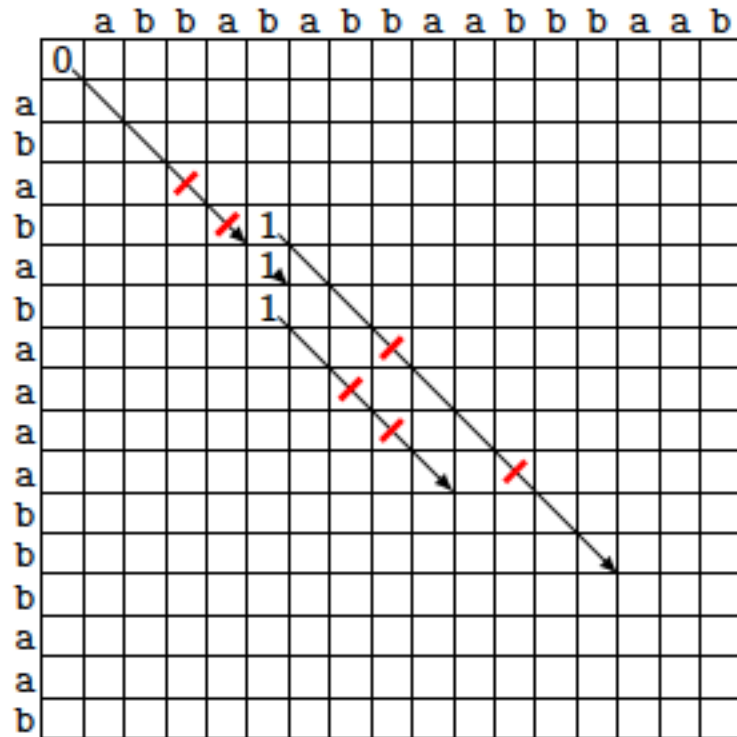
- Reduce the number of positions x with $LCE_{<k}(x, y)$ queries asked.
- For each value i , align all $LCE_{<k}$ queries with the furthest one.
- Cost: $\mathcal{O}(k)$ edits per query.



Algorithm of KS'20

Main idea:

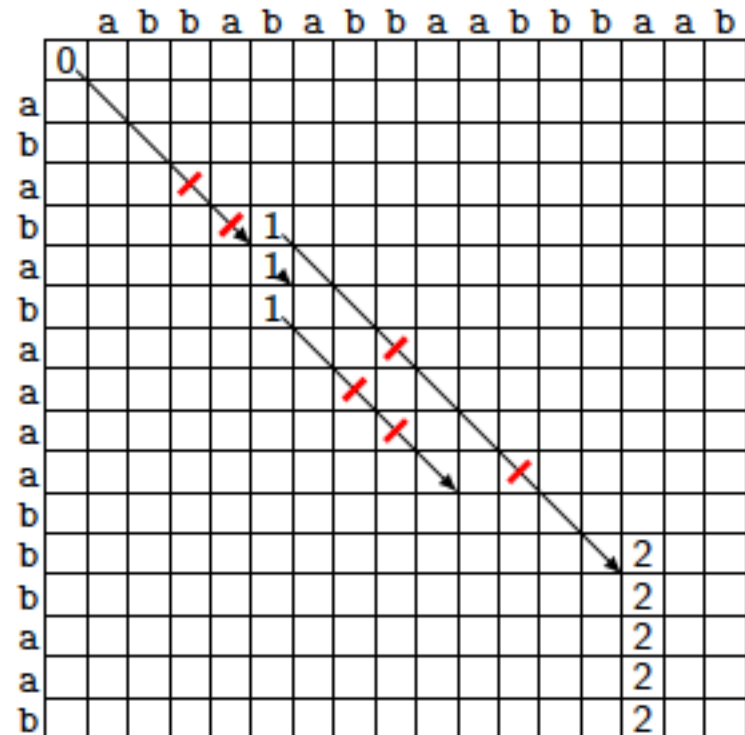
- Reduce the number of positions x with $LCE_{<k}(x, y)$ queries asked.
- For each value i , align all $LCE_{<k}$ queries with the furthest one.
- Cost: $\mathcal{O}(k)$ edits per query.



Algorithm of KS'20

Main idea:

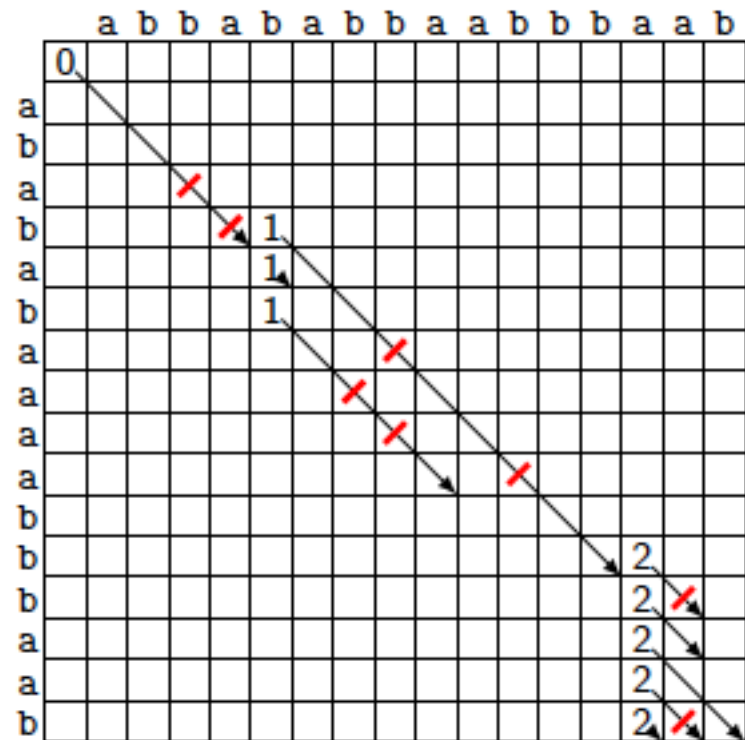
- Reduce the number of positions x with $LCE_{<k}(x, y)$ queries asked.
- For each value i , align all $LCE_{<k}$ queries with the furthest one.
- Cost: $\mathcal{O}(k)$ edits per query.



Algorithm of KS'20

Main idea:

- Reduce the number of positions x with $LCE_{<k}(x, y)$ queries asked.
- For each value i , align all $LCE_{<k}$ queries with the furthest one.
- Cost: $\mathcal{O}(k)$ edits per query.



Algorithm of KS'20

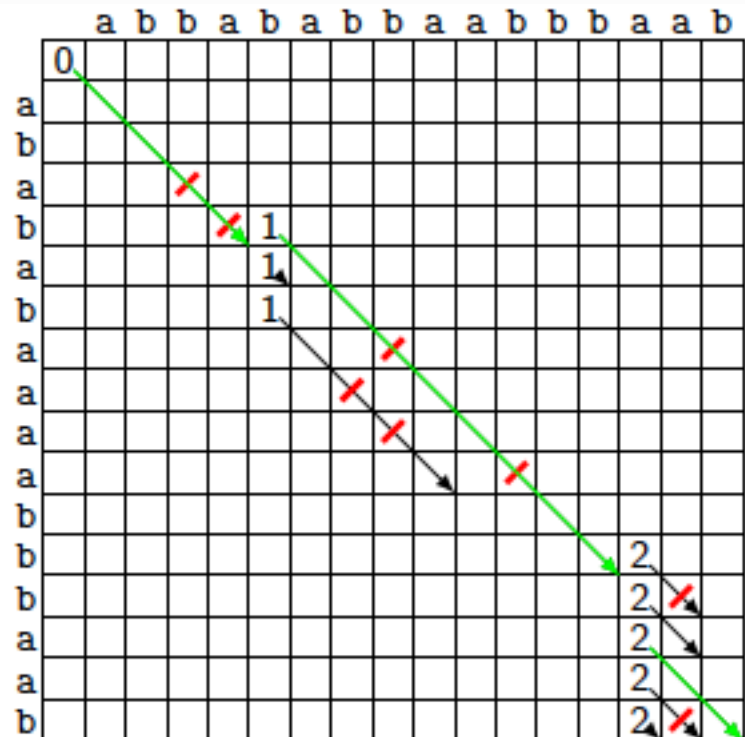
Main idea:

- Reduce the number of positions x with $LCE_{<k}(x, y)$ queries asked.
- For each value i , align all $LCE_{<k}$ queries with the furthest one.
- Cost: $\mathcal{O}(k)$ edits per query.

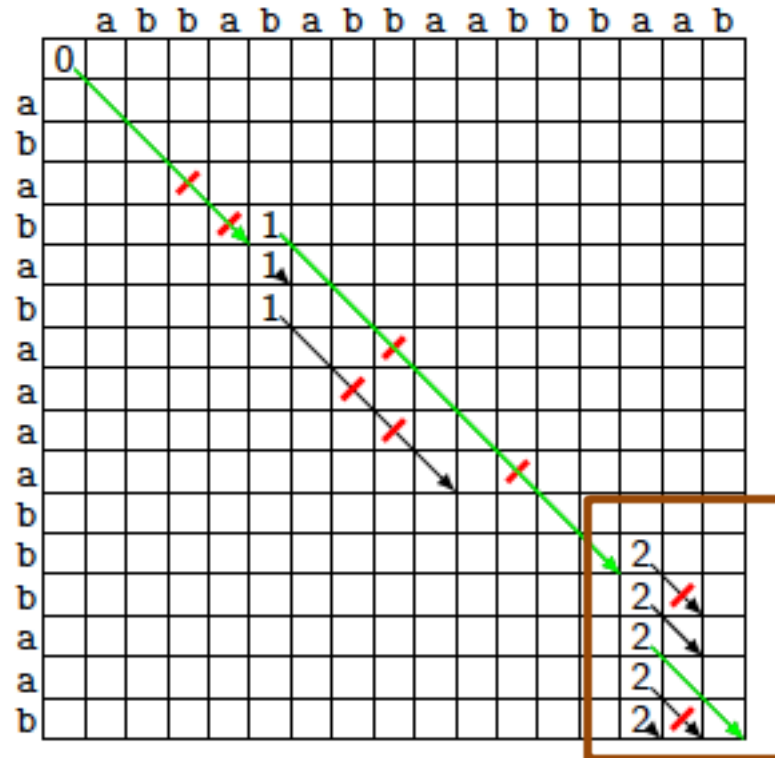
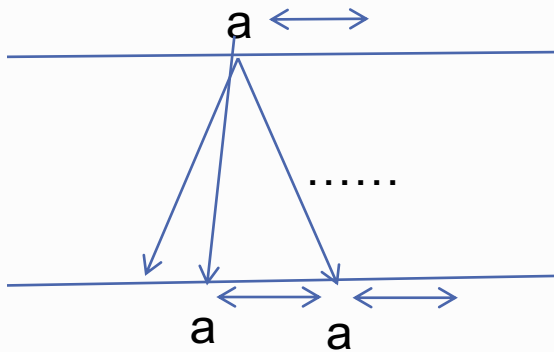
Guarantees:

$ED(X, Y) \leq k \Rightarrow \text{YES}$

$\text{YES} \Rightarrow ED(X, Y) = \mathcal{O}(k^2)$



Algorithm of KS'20

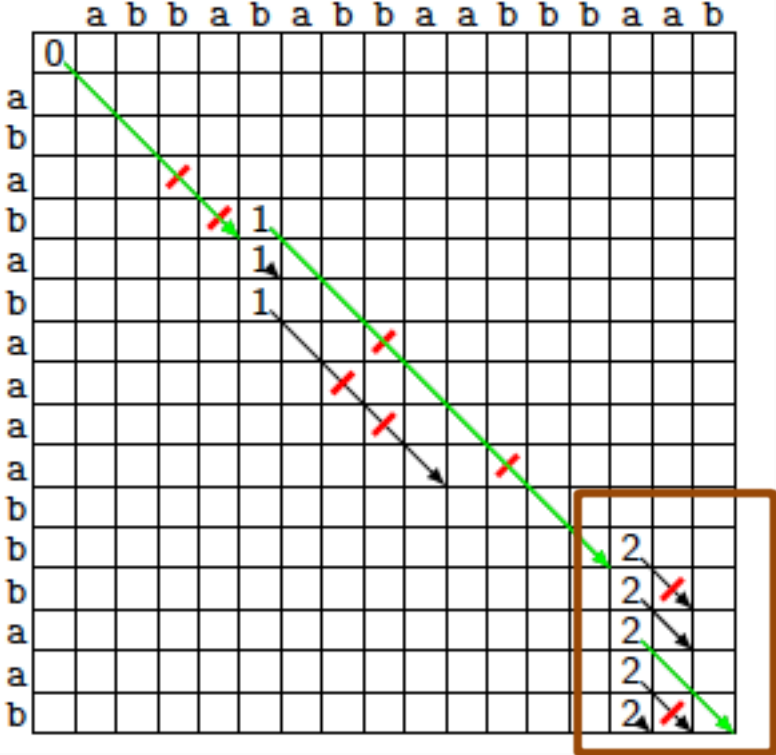
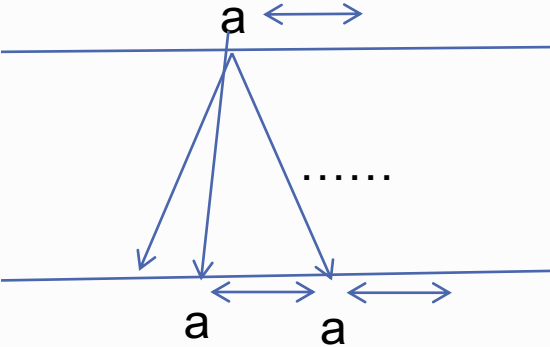


The pattern is periodic.

Instead of checking for each diagonal separately, check for periodicity.



Algorithm of KS'20



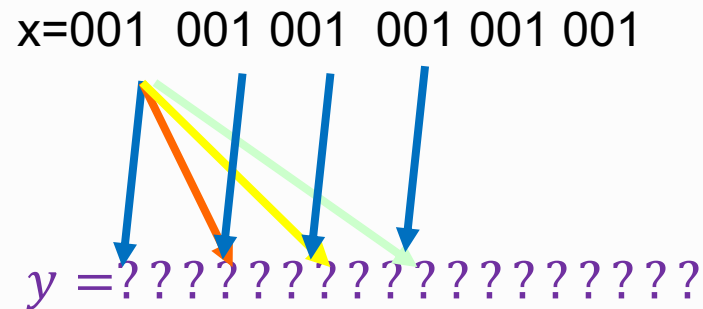
Interestingly, where periodicity breaks, there can be only one possible diagonal to proceed.

Check only that diagonal by sampling.



How Periodicity Helps!

- Suppose x is periodic with period p : $x_i = x_{i+p}$
- The possible matching shifts are p -far apart.

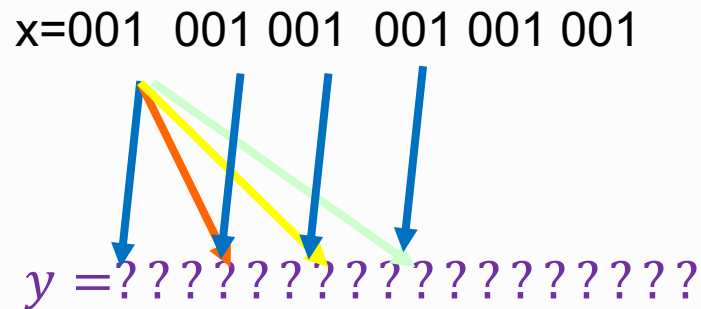


- y is periodic too.



How Periodicity Helps!

- Suppose x is periodic with period p : $x_i = x_{i+p}$
- The possible matching shifts are p -far apart.



- y is periodic too.
- Observation if $p = O(1)$, then there are $\sim k$ shifts to verify

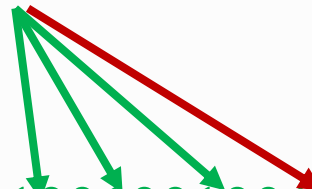


How Periodicity Helps!

- If y does not match the period pattern, it affects the entire set of possible matching shifts except possibly for one.

$x = 001001001001001001001$

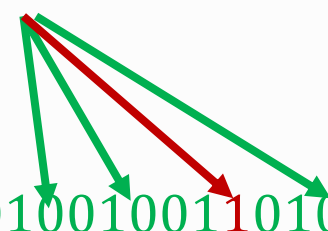
$y = 100100100110100100100$



How Periodicity Helps!

- If y does not match the period pattern, it affects the entire set of possible matching shifts except possibly for one.

$x = 001001001001001001001$
 $y = 100100100110100100100$



The diagram illustrates the alignment of two binary strings, x and y . The string x is `001001001001001001001` and the string y is `100100100110100100100`. A red dot is placed above the first '1' in x and the first '1' in y . Four arrows originate from this red dot: three green arrows point to the first, second, and fourth '1' in y , and one red arrow points to the third '1' in y . This visualizes how a single mismatch in y affects multiple potential shifts.

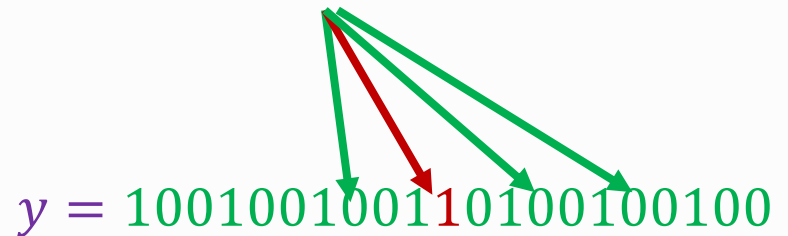


How Periodicity Helps!

- If y does not match the period pattern, it affects the entire set of possible matching shifts except possibly for one.

$x = 001001001001001001001$

$y = 100100100110100100100$



The diagram illustrates the alignment of two binary strings, x and y . The string x is `001001001001001001001` and the string y is `100100100110100100100`. A single green arrow points from the first '1' in x to the first '1' in y . Three other green arrows point from the first '1' in x to the 4th, 8th, and 12th '1's in y . A red arrow points from the first '1' in x to the 7th '1' in y , which is the '1' immediately following the '10' pair in y .

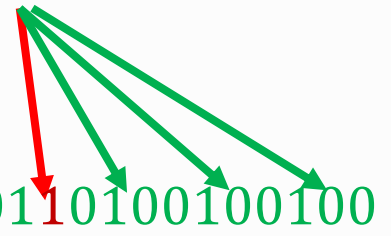


How Periodicity Helps!

- If y does not match the period pattern, it affects the entire set of possible matching shifts except possibly for one.

$x = 001001001001001001001$

$y = 100100100110100100100$



How Periodicity Helps!

- **Key lemma:** When there are multiple possible shifts, we can emulate the comparisons of x_i and y_{i+s} by checking that x, y are still periodic with the same pattern: check $x_i = x_{i-p}$ and $y_i = y_{i-p}$

$x = 001001001001001001$

The string x = 001001001001001001 is shown in purple. Two green curved arrows point from the first '0' to the second '0' and from the second '0' to the third '0', illustrating the periodic nature of the string.

$y = 1001001001101001001$

The string y = 1001001001101001001 is shown in purple. A green curved arrow points from the first '0' to the second '0', and a red curved arrow points from the second '0' to the third '0', illustrating the periodic nature of the string.

- **Observation:** Periodicity can be checked via uniform sampling



Algorithm of KS'20

Main idea:

- Reduce the number of positions x with $\text{LCE}_{<k}(x, y)$ queries asked.
- For each value i , align all $\text{LCE}_{<k}$ queries with the furthest one.
- Cost: $\mathcal{O}(k)$ edits per query.

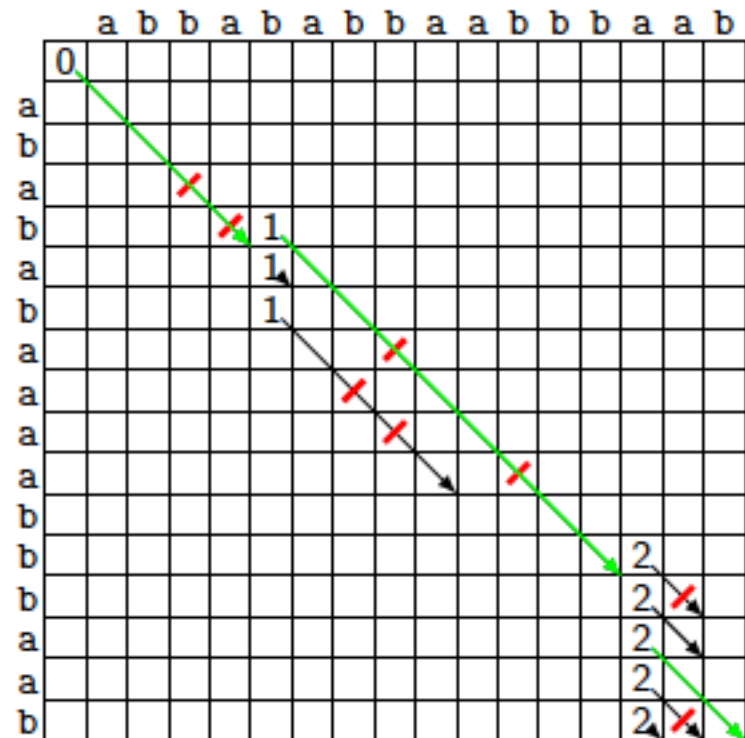
Guarantees:

$\text{ED}(X, Y) \leq k \Rightarrow \text{YES}$

$\text{YES} \Rightarrow \text{ED}(X, Y) = \mathcal{O}(k^2)$

Running time: $\tilde{\mathcal{O}}\left(\frac{n}{k} + k^2\right)$.

Adjust the ideas from GKS19 to process a length- L block in $\tilde{\mathcal{O}}\left(\frac{L}{k} + k\right)$ time.



k vs k' Gap Edit Distance, KS'20

Main idea:

- Set $\alpha = \Theta(\frac{k'}{k})$.
- Use $LCE_{<\alpha}$ queries.
- Align within groups of α diagonals.

	a	b	b	a	b	a	b	b	a	a	b	b	b	a	a	b
a																
b																
a																
b																
a																
b																
a																
a																
a																
b																
b																
b																
a																
a																
b																



k vs k' Gap Edit Distance, KS'20

Main idea:

- Set $\alpha = \Theta(\frac{k'}{k})$.
- Use $LCE_{<\alpha}$ queries.
- Align within groups of α diagonals.

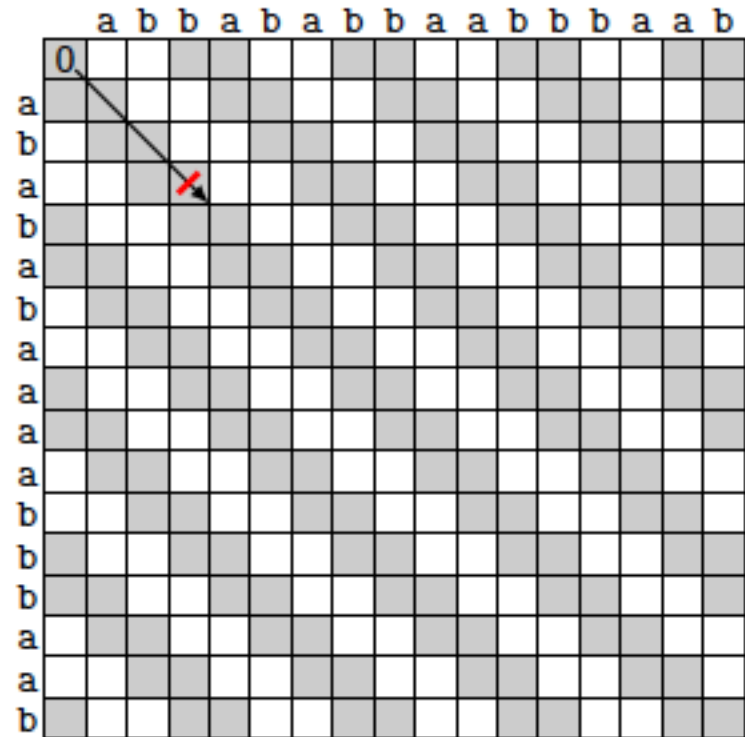
	a	b	b	a	b	a	b	b	a	a	b	b	b	a	a	b	
0																	
a																	
b																	
a																	
b																	
a																	
b																	
a																	
a																	
a																	
b																	
b																	
b																	
a																	
a																	
b																	



k vs k' Gap Edit Distance, KS'20

Main idea:

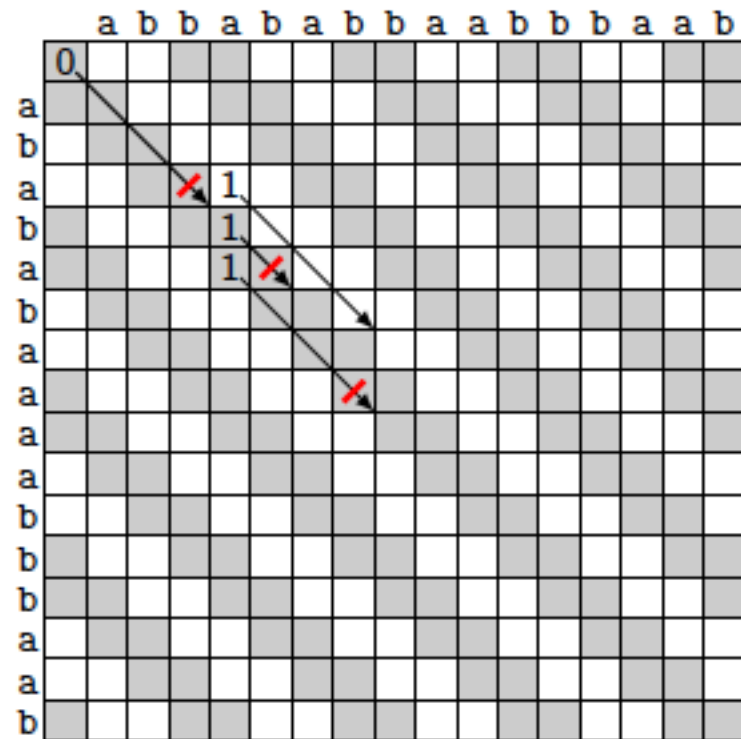
- Set $\alpha = \Theta(\frac{k'}{k})$.
- Use $LCE_{<\alpha}$ queries.
- Align within groups of α diagonals.



k vs k' Gap Edit Distance, KS'20

Main idea:

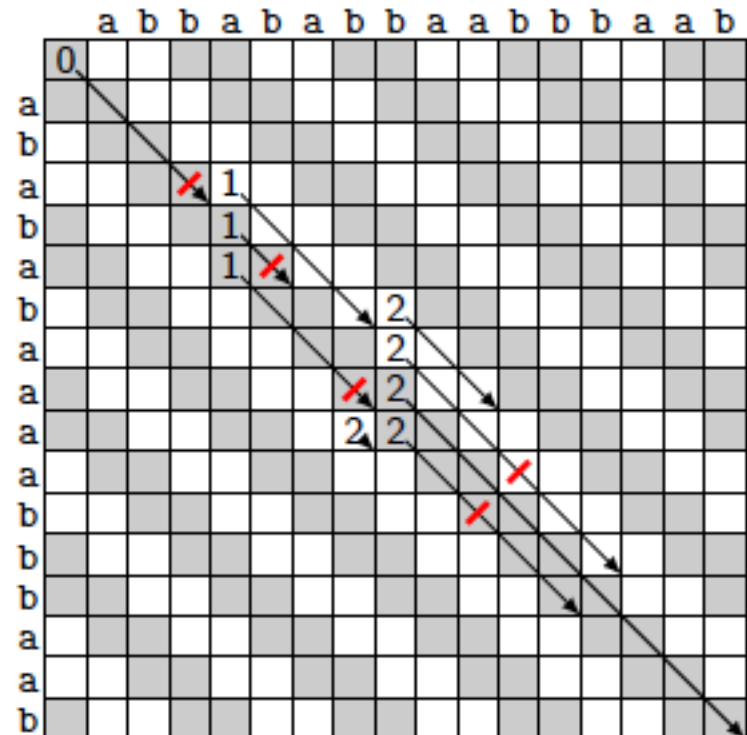
- Set $\alpha = \Theta(\frac{k'}{k})$.
- Use $LCE_{<\alpha}$ queries.
- Align within groups of α diagonals.



k vs k' Gap Edit Distance, KS'20

Main idea:

- Set $\alpha = \Theta(\frac{k'}{k})$.
- Use $LCE_{<\alpha}$ queries.
- Align within groups of α diagonals.



k vs k' Gap Edit Distance, KS'20

Main idea:

- Set $\alpha = \Theta\left(\frac{k'}{k}\right)$.
- Use $\text{LCE}_{<\alpha}$ queries.
- Align within groups of α diagonals.

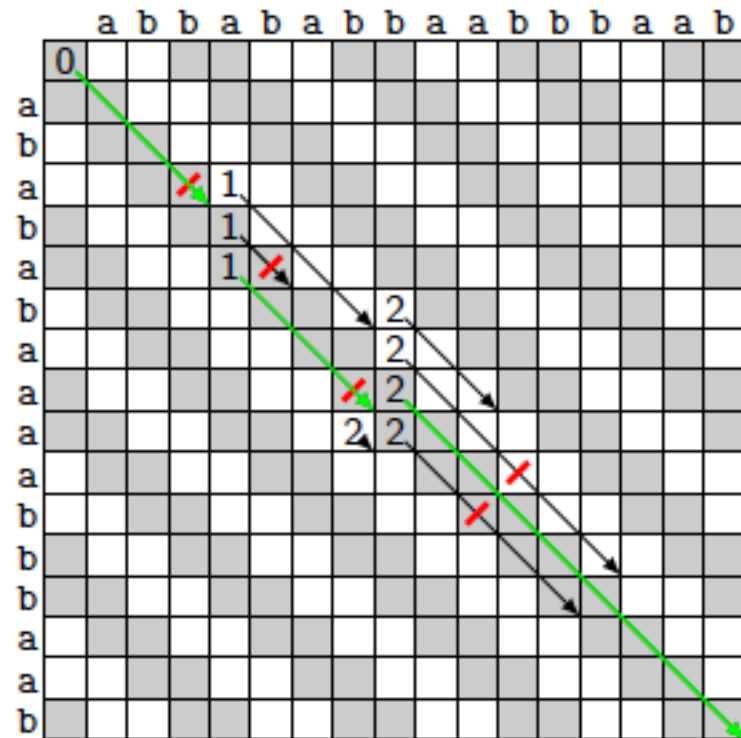
Guarantees:

$\text{ED}(X, Y) \leq k \Rightarrow \text{YES}$

$\text{YES} \Rightarrow \text{ED}(X, Y) = \mathcal{O}(k\alpha)$

Simple implementation: $\tilde{\mathcal{O}}\left(\frac{n+k^3}{\alpha}\right)$ time.

With some more tricks $\tilde{\mathcal{O}}\left(\frac{n}{\alpha} + k^2 + \frac{\sqrt{nk^3}}{\alpha}\right)$.



Is rough computation good enough?

- Favorite applications: Bioinformatics
 - But DNA is really long!!

- Many pairs of DNA sequences may need to be compared- if we know pair-wise distance is small, we want to compute it exactly in sublinear time.



Sublinear Time Edit Distance with Preprocessing [GRS'20]

- We have to pay linear time to sequence a DNA anyway!



Edit Distance with Preprocessing

Preprocessing:

Preprocess each string separately



Alice the Ocelot

$$A \rightarrow f(A)$$

$$B \rightarrow g(B)$$



Bob the Bobcat

Query:

Given two strings and preprocess output, compute edit distance

$$A, f(A), B, g(B) \rightarrow ED(A, B)$$



Can Preprocessing Help?

- NO, in the worst case!
 - Assuming SETH, no subquadratic exact algorithm for edit distance with polynomial preprocessing time
- YES, in reality!
 - Preprocessing $O(n \log n)$
 - Query time $O(k^2)$: returns exact answer!!



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $O(n + k^2)$ time.

DP table:

$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x + 1][y + 1] \in \{D[x][y], D[x][y] + 1\}$.

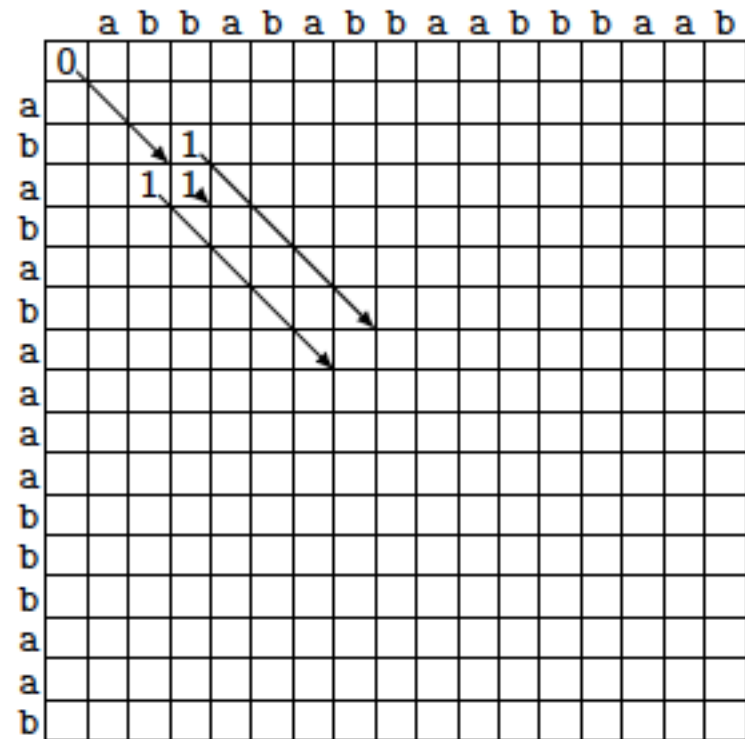
Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

- 1 Identify the furthest cell with an $(i - 1)$ -valued neighbor.
- 2 Proceed forward using an LCE query.

Preprocessing is done using suffix tree by appending string y to x : $x\$y$

LCE Query: (i, j) : find max d
 $x[i, \dots, i+d] = y[j, \dots, j+d]$

Query time: $O(1)$



Algorithm of Landau & Vishkin 1998

Test $ED(X, Y) \leq k$ in $O(n + k^2)$ time.

DP table:

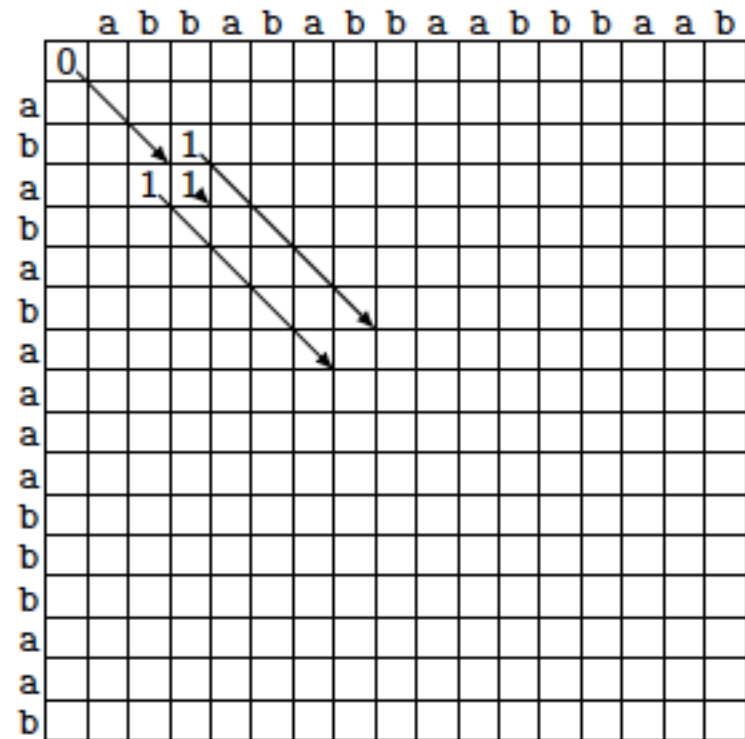
$D[x][y] = ED(X[0..x], Y[0..y])$.

Observation:

$D[x+1][y+1] \in \{D[x][y], D[x][y] + 1\}$.

Algorithm: For $i = 0, \dots, k$, compute the furthest i -valued cell on each diagonal.

- 1 Identify the furthest cell with an $(i - 1)$ -valued neighbor.
- 2 Proceed forward using an LCE query.



Preprocess each string separately using rolling hash of windows of size 2^l for $l=1,2,\dots,\log n$

LCE Query: (i,j) : find $\max d$ $x[i,\dots,i+d]=y[j,\dots,j+d]$

Query time: $O(\log n)$



Open Questions & Progress

- **Edit Distance Computation**
 - **Sublinear Time:** Lower bound & better adaptive algorithms [Further progress has happened]
 - **Dynamic Algorithm:** Improving over trivial bounds
 - E.g., k vs k^2 in $\Theta(k)$ update time is possible. Can we do better? [See our upcoming paper in FOCS'23]
 - **Near-neighbor search**
 - **Approximation Algorithms:** Subquadratic approximation scheme
 - **Beyond Worst Case** [Further progress has happened for multi strings, see our paper in APPROX'22]
- **Generalizations & Other Sequence Similarity Measures**
 - Longest Increasing Subsequence
 - Longest Common Subsequence
 - Model to Data distance: Language Edit Distance
 - Exploit Lipschitz property
- **Graph Distances**
 - Tree Edit Distance
- **Computing over Compressed Data**
 - Lossless vs Lossy [See our paper in SODA'22]

