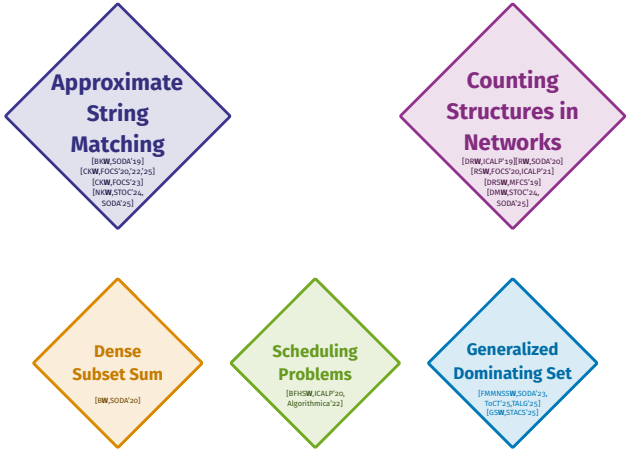
Revitalizing Research on Approximate String Matching Algorithms

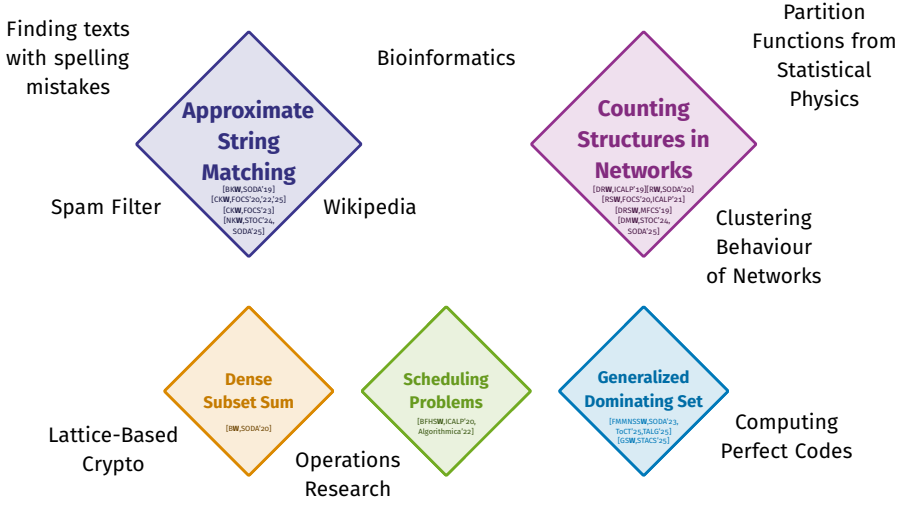
Philip Wellnitz
National Institute of Informatics

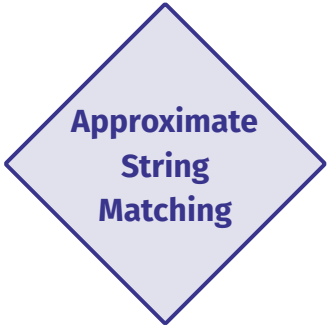
Based on joint works with Karl Bringmann, Alejandro Cassis, Panagiotis Charalampopoulos, Tomasz Kociumaka, Marvin Künnemann, and Jakob Nogler.

Research Focus: Theoretical Guarantees for Fundamental Problems



Research Focus: Theoretical Guarantees for Fundamental Problems of Practical Relevance




Approximate
String
Matching

An Example

...

(TU Kaiserslautern).

(Simons Institute and UC Berkeley), (University of Salzburg), (University of Salzburg).

(University of Pennsylvania), (University of Pennsylvania), (University of Pennsylvania).

(Reichman University, Herzliya, Israel and Birkbeck, University of London), (UC Berkeley and Max Planck Institute for Informatics, SIC, Saarbrücken, Germany), (Max Planck Institute for Informatics, SIC, Saarbrücken, Germany).

(National Institute of Informatics).

(UniversitÃ© Paris CitÃ©, CNRS, IRIF, F-75013, Paris, France); (Universitat PolitÃ¨cnica de Catalunya)

...

(ETH Zurich); (Toyota Technological Institute at Chicago)

(Merton College, University of Oxford, United Kingdom); (Mathematical Institute, University of Bonn, Germany); (Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbr|cken, Germany)

(The Academic College of Tel Aviv-Yafo), (UC Berkeley), (Weizmann Institute of Science), (University of California San Diego).

(CISPA Helmholtz Center for Information Security, Saarbrücken); (Simon Fraser University); (Duke University); (CISPA Helmholtz Center for Information Security, Saarbrücken); (Max Planck Institute for Informatics, SIC, Saarbrücken)

...

(Alibaba Group); (Microsoft Research); (Washington University in St. Louis); (Columbia University); (University of Chicago)

(Stony Brook University); (Max Planck Institute for Informatics, Saarbruecken)

(Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, France); (University of Warsaw, Poland); (Saarland University and Max Planck Institute for Informatics, SaarbrÃ¼cken, Germany); (University of Warsaw, Poland)

(ETH ZÃ¼rich); (TU Berlin)

(Carnegie Mellon University)

...

An Example

...

(TU Kaiserslautern).

(Simons Institute and UC Berkeley), (University of Salzburg), (University of Salzburg).

(University of Pennsylvania), (University of Pennsylvania), (University of Pennsylvania).

(Reichman University, Herzliya, Israel and Birkbeck, University of London), (UC Berkeley and Max Planck Institute for Informatics, SIC, **Saarbrücken**, Germany), (Max Planck Institute for Informatics, SIC, **Saarbrücken**, Germany).

(National Institute of Informatics).

(UniversitÃ© Paris CitÃ©, CNRS, IRIF, F-75013, Paris, France); (Universitat PolitÃ¨cnica de Catalunya)

...

(ETH Zurich); (Toyota Technological Institute at Chicago)

(Merton College, University of Oxford, United Kingdom); (Mathematical Institute, University of Bonn, Germany); (Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), Saarbr|cken, Germany)

(The Academic College of Tel Aviv-Yafo), (UC Berkeley), (Weizmann Institute of Science), (University of California San Diego).

(CISPA Helmholtz Center for Information Security, **Saarbrücken**); (Simon Fraser University); (Duke University); (CISPA Helmholtz Center for Information Security, **Saarbrücken**); (Max Planck Institute for Informatics, SIC, **Saarbrücken**)

...

(Alibaba Group); (Microsoft Research); (Washington University in St. Louis); (Columbia University); (University of Chicago)

(Stony Brook University); (Max Planck Institute for Informatics, Saarbruecken)

(Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, France); (University of Warsaw, Poland); (Saarland University and Max Planck Institute for Informatics, SaarbrÃ¼cken, Germany); (University of Warsaw, Poland)

(ETH ZÃ¼rich); (TU Berlin)

(Carnegie Mellon University)

...

An Example

...

(TU Kaiserslautern).

(Simons Institute and UC Berkeley), (University of Salzburg), (University of Salzburg).

(University of Pennsylvania), (University of Pennsylvania), (University of Pennsylvania).

(Reichman University, Herzliya, Israel and Birkbeck, University of London), (UC Berkeley and Max Planck Institute for Informatics, SIC, **Saarbrücken**, Germany), (Max Planck Institute for Informatics, SIC, **Saarbrücken**, Germany).

(National Institute of Informatics).

(UniversitÃ© Paris CitÃ©, CNRS, IRIF, F-75013, Paris, France); (Universitat PolitÃ¯cnica de Catalunya)

...

(ETH Zurich); (Toyota Technological Institute at Chicago)

(Merton College, University of Oxford, United Kingdom); (Mathematical Institute, University of Bonn, Germany); (Max Planck Institute for Informatics, Saarland Informatics Campus (SIC), **Saarbr|cken**, Germany)

(The Academic College of Tel Aviv-Yafo), (UC Berkeley), (Weizmann Institute of Science), (University of California San Diego).

(CISPA Helmholtz Center for Information Security, **Saarbrücken**); (Simon Fraser University); (Duke University); (CISPA Helmholtz Center for Information Security, **Saarbrücken**); (Max Planck Institute for Informatics, SIC, **Saarbrücken**)

...

(Alibaba Group); (Microsoft Research); (Washington University in St. Louis); (Columbia University); (University of Chicago)

(Stony Brook University); (Max Planck Institute for Informatics, **Saarbruecken**)

(Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, France); (University of Warsaw, Poland); (Saarland University and Max Planck Institute for Informatics, **SaarbrÃ¼cken**, Germany); (University of Warsaw, Poland)

(ETH ZÃ¼rich); (TU Berlin)

(Carnegie Mellon University)

...

Task: Find **Saarbrücken** in a text.

Task: Find **Saarbrücken** in a text.

Or Saarbruecken.

Task: Find **Saarbrücken** in a text.

Or Saarbruecken. Or Sarrebruck.

Task: Find **Saarbrücken** in a text.

Or Saarbruecken. Or Sarrebruck. Or Saarbrucken, Saarbr|cken, SaarbrÃ¼cken, ...

early 1980's

Approximate String Matching

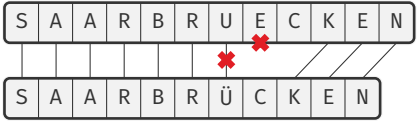
Given a text T, a pattern P, and an integer k, identify the (starting positions of) substrings of T that are at **edit distance** of at most k to P.

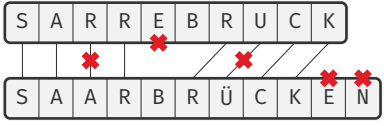
early 1980's

Approximate String Matching

Given a text T, a pattern P, and an integer k, identify the (starting positions of) substrings of T that are at **edit distance** of at most k to P.

Edit distance: minimum number of insertions, deletions, or substitutions of single characters to transform one string into another string





Outline

Basic Tricks and Tools, Previous Work

New Algorithms via Structural Insights

Spotlight Extension: Quantum Algorithms for ASM

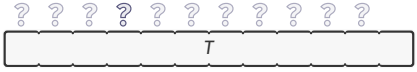
Spotlight Extension: String Matching with Weighted Edits

Open Problems and Future Directions

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.



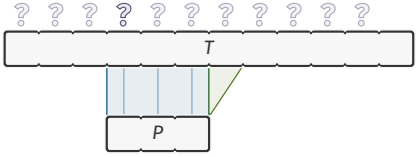
Focus: Obtain starting positions of occurrences

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

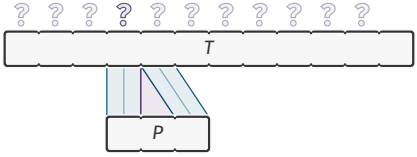


Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

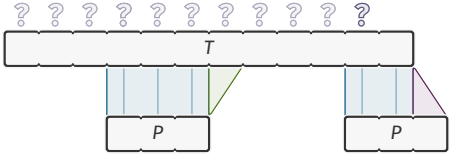


Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences



Approximate String Matching

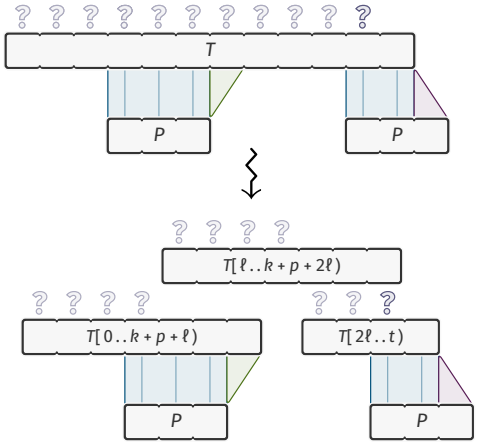
early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Standard Trick": write $t := |T|$, $p := |P|$
Split T into overlapping fragments of len $\ell + p + k$

$\rightsquigarrow O(t/\ell)$ instances,
each "responsible" for its first ℓ positions



Approximate String Matching

early 1980's

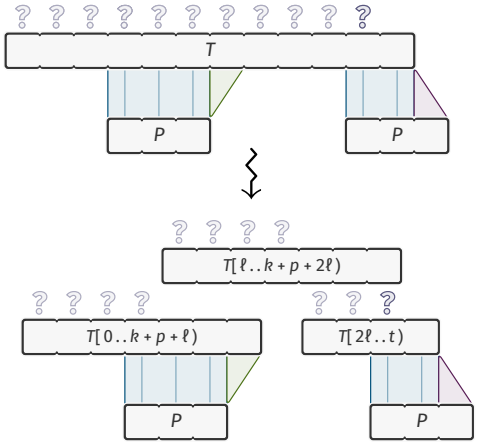
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Standard Trick": write $t := |T|$, $p := |P|$
Split T into overlapping fragments of len $\ell + p + k$

$\rightsquigarrow O(t/\ell)$ instances,
each "responsible" for its first ℓ positions

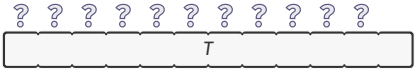
$\rightsquigarrow$ Useful special cases: $\ell = k$ and $\ell = 0.5\,p$



Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.



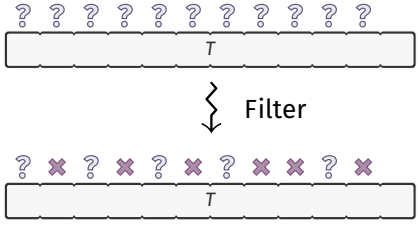
Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.



Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Approximate String Matching

early 1980's

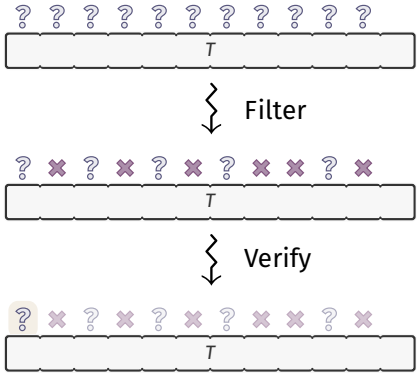
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position



Approximate String Matching

early 1980's

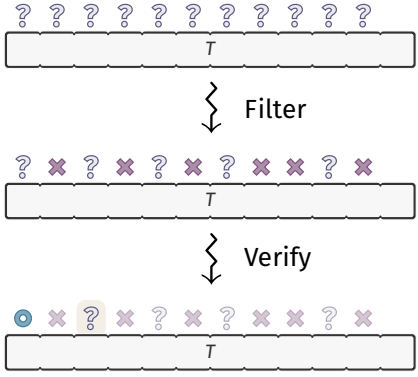
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position



Approximate String Matching

early 1980's

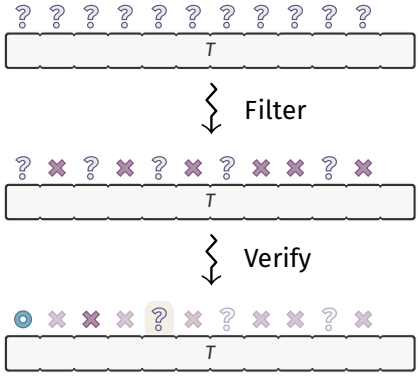
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position



Approximate String Matching

early 1980's

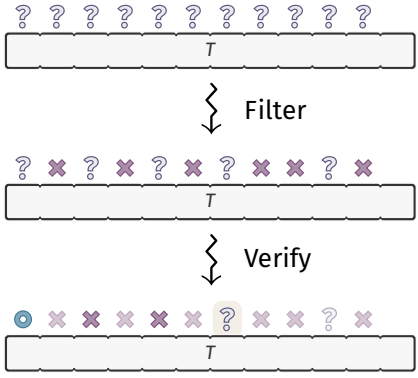
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position



Approximate String Matching

early 1980's

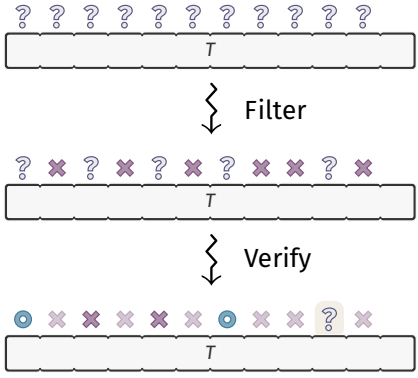
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position



Approximate String Matching

early 1980's

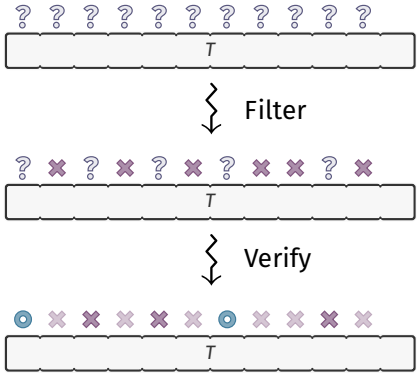
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2, Verify: (typically slow)
Check for occ at each remaining position



Approximate String Matching

early 1980's

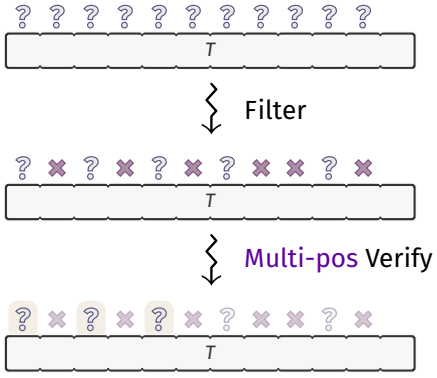
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2', Multi-pos Verify: (typically slow)
Check for occ at each remaining position, multiple positions at once



Approximate String Matching

early 1980's

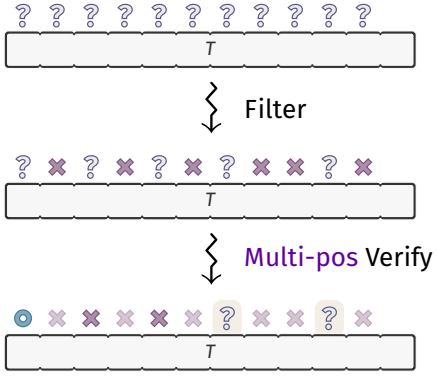
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2', Multi-pos Verify: (typically slow)
Check for occ at each remaining position, multiple positions at once



Approximate String Matching

early 1980's

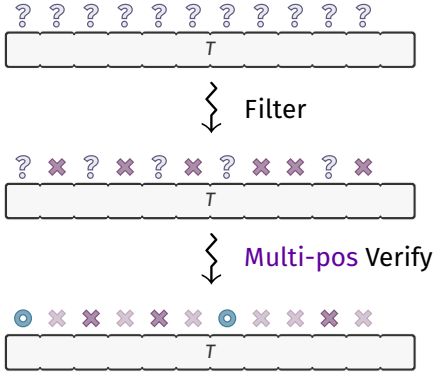
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Focus: Obtain starting positions of occurrences

"Filter and Verify Paradigm"

Step 1, Filter: (typically fast)
Compute (small) superset of starting positions

Step 2', Multi-pos Verify: (typically slow)
Check for occ at each remaining position, multiple positions at once

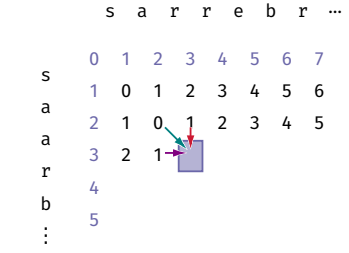


early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm	($t := \|T\|, p := \|P\|$)
Textbook DP $\rightsquigarrow O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow O(t^2 p)$	substr



$e_{i,0} = i$ (del $T[0..i)$); $e_{0,j} = j$ (ins $P[0..j)$)

$$e_{i,j} = \min \begin{cases} e_{i-1,j} + 1 & \text{(del from } T\text{)} \\ e_{i,j-1} + 1 & \text{(ins in } T\text{)} \\ e_{i-1,j-1} + (T[i] \neq P[j]) & \text{(match/subst)} \end{cases}$$

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm	($t := \|T\|, p := \|P\|$)
Textbook DP $\rightsquigarrow O(tp)$	Verify pos 1 by 1 $\rightsquigarrow O(t^2 p)$	substr
[Sellers, 1980] and others	Verify all pos at once $\rightsquigarrow O(tp)$	start pos

		r	b	e	r	r	a	s	…
	0	0	0	0	0	0	0	0	
b	1	1	0	1	1	1	1	1	
r	2	1	1	1	1	1	2	2	
a	3	2	2	2	2	2	1	2	
a	4	3	3	3	3	3	2	2	
s	5	4	4	4	4	4	3	2	

$e_{i,0} = 0$ (del $T[0..i)$); $e_{0,j} = j$ (ins $P[0..j)$)

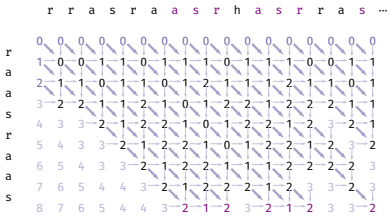
$$e_{i,j} = \min \begin{cases} e_{i-1,j} + 1 & \text{(del from } T\text{)} \\ e_{i,j-1} + 1 & \text{(ins in } T\text{)} \\ e_{i-1,j-1} + (T[i] \neq P[j]) & \text{(match/subst)} \end{cases}$$

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm	($t := \|T\|, p := \|P\|$)
Textbook DP $\rightsquigarrow O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow O(t^2 p)$ Verify all pos at once $\rightsquigarrow O(tp)$	substr start pos
DP, diagonally compute only entries $\leq k$ $\rightsquigarrow O(t + kp)$	Verify pos 1 by 1 $\rightsquigarrow O(tkp)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)p) = O(tp)$	substr start pos



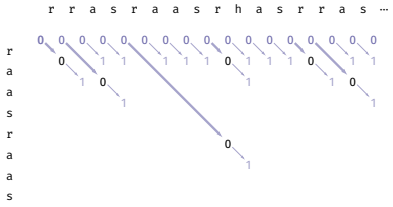
Prune whenever $e_{i,j} > k$.

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm ($t := \|T\|, p := \|P\|$)	
Textbook DP $\rightsquigarrow O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow O(t^2 p)$ Verify all pos at once $\rightsquigarrow O(tp)$	substr start pos
DP, diagonally compute only entries $\leq k$ $\rightsquigarrow O(t + kp)$	Verify pos 1 by 1 $\rightsquigarrow O(tkp)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)p) = O(tp)$	substr start pos
compute DP diagonals, skip over common substr in $O(1)$ $\rightsquigarrow O(t + k^2)$ [Landau, Vishkin'89].	Verify pos 1 by 1 $\rightsquigarrow O(tk^2)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)k) = O(tk)$	substr start pos



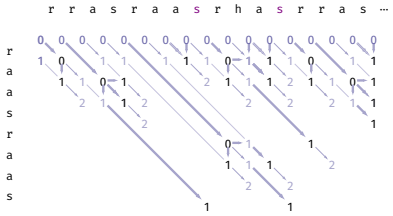
Comp. furthest pos on diag d reachable w/ $\ell = 0, \ldots, k$ edits.
$\rightsquigarrow$ Jump over equal substrings in $O(1)$ time

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm ($t := \|T\|, p := \|P\|$)	
Textbook DP $\rightsquigarrow O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow O(t^2p)$ Verify all pos at once $\rightsquigarrow O(tp)$	substr start pos
DP, diagonally compute only entries $\leq k$ $\rightsquigarrow O(t + kp)$	Verify pos 1 by 1 $\rightsquigarrow O(tkp)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)p) = O(tp)$	substr start pos
compute DP diagonals, skip over common substr in $O(1)$ $\rightsquigarrow O(t + k^2)$ [Landau, Vishkin'89].	Verify pos 1 by 1 $\rightsquigarrow O(tk^2)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)k) = O(tk)$	substr start pos



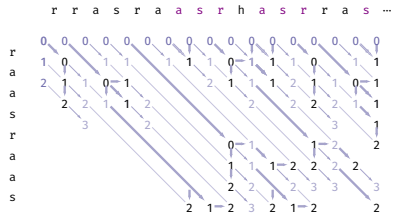
Comp. furthest pos on diag d reachable w/ $\ell = 0, \ldots, k$ edits.
$\rightsquigarrow$ Jump over equal substrings in $O(1)$ time

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm ($t := \|T\|, p := \|P\|$)	
Textbook DP $\rightsquigarrow O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow O(t^2p)$ Verify all pos at once $\rightsquigarrow O(tp)$	substr start pos
DP, diagonally compute only entries $\leq k$ $\rightsquigarrow O(t + kp)$	Verify pos 1 by 1 $\rightsquigarrow O(tkp)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)p) = O(tp)$	substr start pos
compute DP diagonals, skip over common substr in $O(1)$ $\rightsquigarrow O(t + k^2)$ [Landau, Vishkin'89].	Verify pos 1 by 1 $\rightsquigarrow O(tk^2)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)k) = O(tk)$	substr start pos



Comp. furthest pos on diag d reachable w/ $\ell = 0, \ldots, k$ edits.
$\rightsquigarrow$ Jump over equal substrings in $O(1)$ time

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Edit dist/Verify Algorithm	ASM Algorithm ($t := \|T\|, p := \|P\|$)	
Textbook DP $\rightsquigarrow O(tp)$ [Sellers, 1980] and others	Verify pos 1 by 1 $\rightsquigarrow O(t^2 p)$ Verify all pos at once $\rightsquigarrow O(tp)$	substr start pos
DP, diagonally compute only entries $\leq k$ $\rightsquigarrow O(t + kp)$	Verify pos 1 by 1 $\rightsquigarrow O(tkp)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)p) = O(tp)$	substr start pos
compute DP diagonals, skip over common substr in $O(1)$ $\rightsquigarrow O(t + k^2)$ [Landau, Vishkin'89].	Verify pos 1 by 1 $\rightsquigarrow O(tk^2)$ Verify $\ell \leq t$ pos at once $\rightsquigarrow O(t/\ell \cdot (\ell + k)k) = O(tk)$	substr start pos

$\rightsquigarrow$ Didn't use filter yet...

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

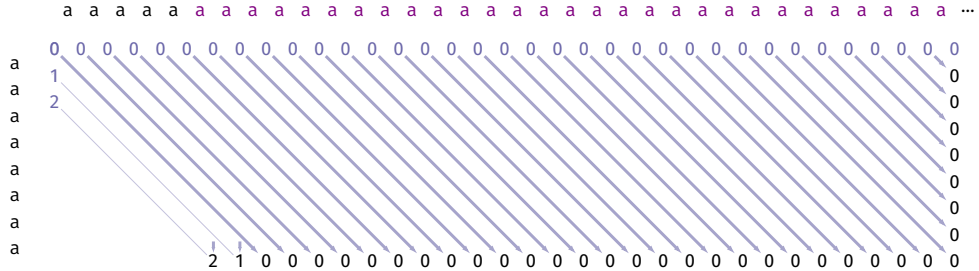
Idea: Can we filter out diagonals that will never lead to an occurrence?

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Idea: Can we filter out diagonals that will never lead to an occurrence?

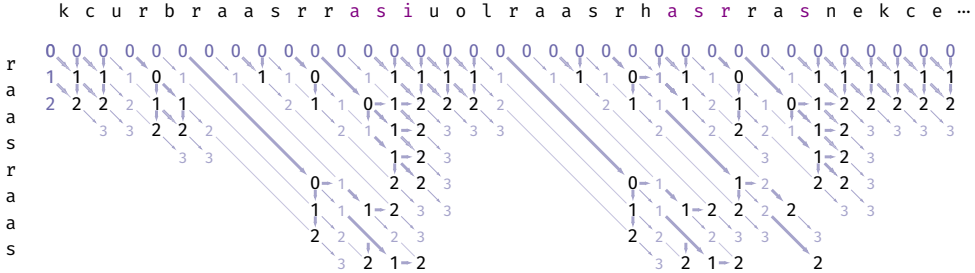


Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Idea: Can we filter out diagonals that will never lead to an occurrence?
Sometimes we can; and if we cannot, there might be structure to exploit!

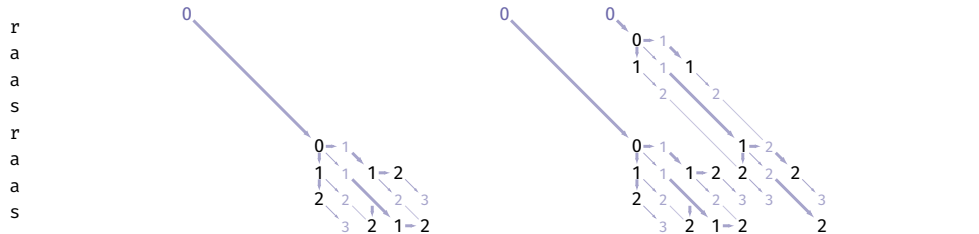


early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

Idea: Can we filter out diagonals that will never lead to an occurrence?
Sometimes we can; and if we cannot, there might be structure to exploit!



Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

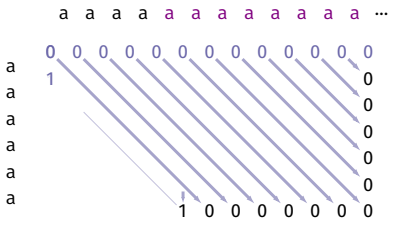
- Idea: Find parts of P that are rare in T
  - $\rightsquigarrow$ Can filter out candidates for occurrences

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- Idea: Find parts of P that are rare in T
  - ⇝ Can filter out candidates for occurrences
- Seen: Highly repetitive parts R are bad...
  - ⇝ R may appear $\approx t$ times in T

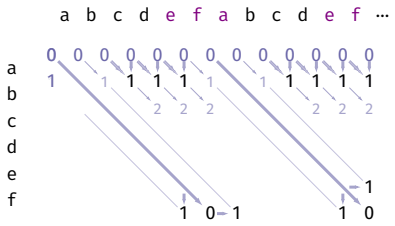


early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- Idea: Find parts of P that are rare in T
  - ⇝ Can filter out candidates for occurrences
- Seen: Highly repetitive parts R are bad...
  - ⇝ R may appear $\approx t$ times in T
- Non-repetitive parts B are good!
  - ⇝ B may appear only $\approx t/b$ times in T

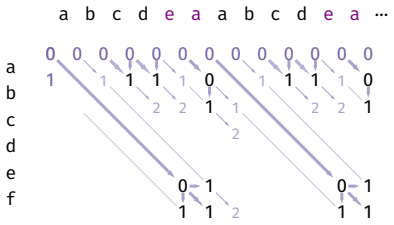


early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- Idea: Find parts of P that are rare in T
  - ⇝ Can filter out candidates for occurrences
- Seen: Highly repetitive parts R are bad...
  - ⇝ R may appear $\approx t$ times in T
- Non-repetitive parts B are good!
  - ⇝ B may appear only $\approx t/b$ times in T
  - ⇝ Problem: B may appear approximately in T

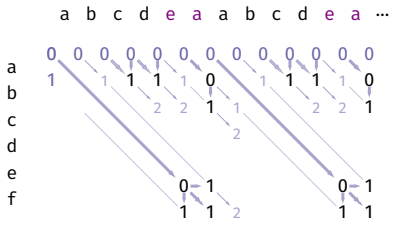


Approximate String Matching

early 1980's

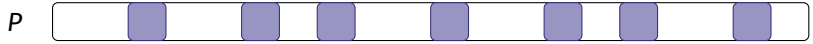
Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- Idea: Find parts of P that are rare in T
  - ⇝ Can filter out candidates for occurrences
- Seen: Highly repetitive parts R are bad...
  - ⇝ R may appear $\approx t$ times in T
- Non-repetitive parts B are good!
  - ⇝ B may appear only $\approx t/b$ times in T
  - ⇝ Problem: B may appear approximately in T
  - ⇝ Use $> k$ disjoint breaks



Suppose we have: $2k$ disjoint breaks $B_1, \ldots, B_{2k}$ in P such that

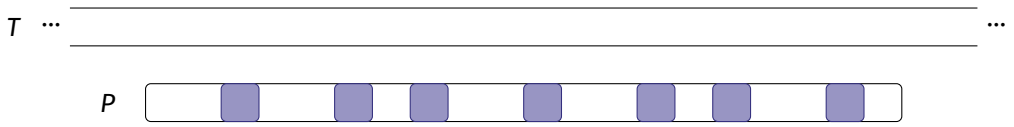
- $|B_i| = \Theta(p/k)$ and
- $|B_i|$ is not periodic



Suppose we have: $2k$ disjoint breaks $B_1, \ldots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- $|B_i|$ is not periodic

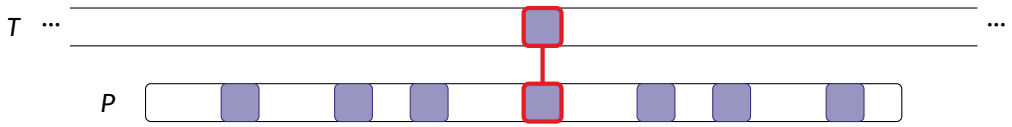
- In any k-edit occ, at least 1 break is matched exactly. (budget for edits is k)



Suppose we have: $2k$ disjoint breaks $B_1, \ldots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- $|B_i|$ is not periodic

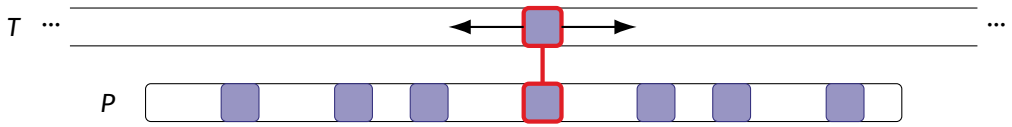
- In any k-edit occ, at least 1 break is matched exactly. (budget for edits is k)
- Algorithm idea: For each B_i, find exact matches of B_i in T



Suppose we have: $2k$ disjoint breaks $B_1, \ldots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- $|B_i|$ is not periodic

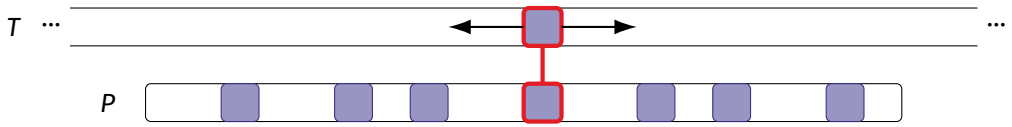
- In any k-edit occ, at least 1 break is matched exactly. (budget for edits is k)
- Algorithm idea: For each B_i, find exact matches of B_i in T,
  Try to extend each exact match into an occ using $O(n + k^2)$)-time Verify [LV'89]



Suppose we have: $2k$ disjoint breaks $B_1, \ldots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- $|B_i|$ is not periodic

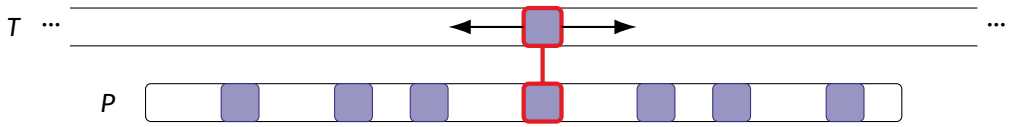
- In any k-edit occ, at least 1 break is matched exactly. (budget for edits is k)
- Algorithm idea: For each B_i, find exact matches of B_i in T,
  Try to extend each exact match into an occ using $O(n + k^2)$)-time Verify [LV'89]
- Have at most $t/(|B_i|/2) = \Theta(kt/p)$ exact occ's of B_i



Suppose we have: $2k$ disjoint breaks $B_1, \ldots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- $|B_i|$ is not periodic

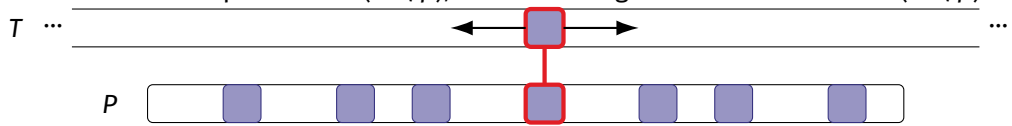
- In any k-edit occ, at least 1 break is matched exactly. (budget for edits is k)
- Algorithm idea: For each B_i, find exact matches of B_i in T,
  Try to extend each exact match into an occ using $O(n + k^2)$)-time Verify [LV'89]
- Have at most $t/(|B_i|/2) = \Theta(kt/p)$ exact occ's of B_i
  $\rightsquigarrow$ $O(k^2 t/p)$ calls to [LV'89]; $O(k^4 t/p)$ time in total



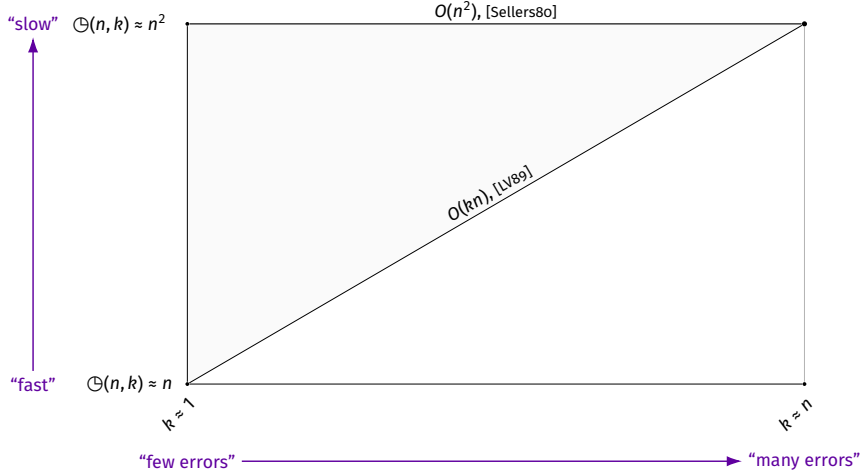
Suppose we have: $2k$ disjoint breaks $B_1, \ldots, B_{2k}$ in P such that

- $|B_i| = \Theta(p/k)$ and
- $|B_i|$ is not periodic

- In any k-edit occ, at least k breaks are matched exactly. (budget for edits is k)
- Algorithm idea: For each B_i, find exact matches of B_i in T,
  Try to extend each exact match into an occ using $O(n + k^2)$)-time Verify [LV'89]
- Have at most $t/(|B_i|/2) = \Theta(kt/p)$ exact occ's of B_i
  $\rightsquigarrow$ $O(k^2 t/p)$ calls to [LV'89]; $O(k^4 t/p)$ time in total
- Can even be improved to $O(k^3 t/p)$; but not-enough breaks case is also $O(k^4 t/p)$.

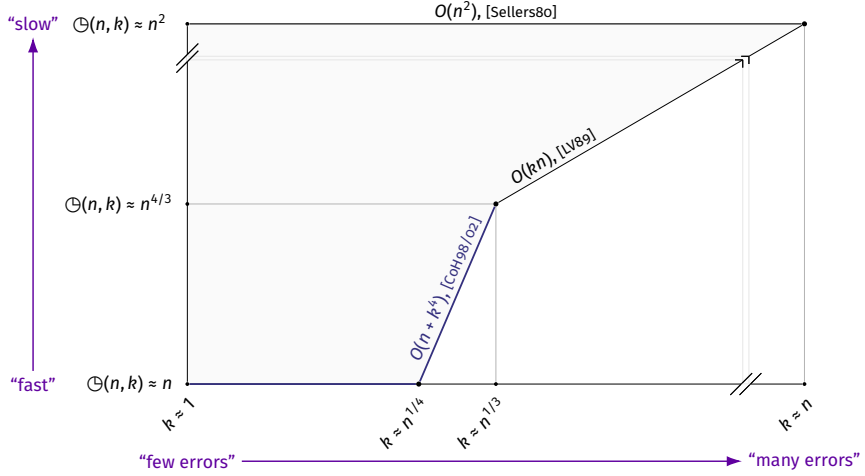


Recap: State-of-the-Art Algorithms



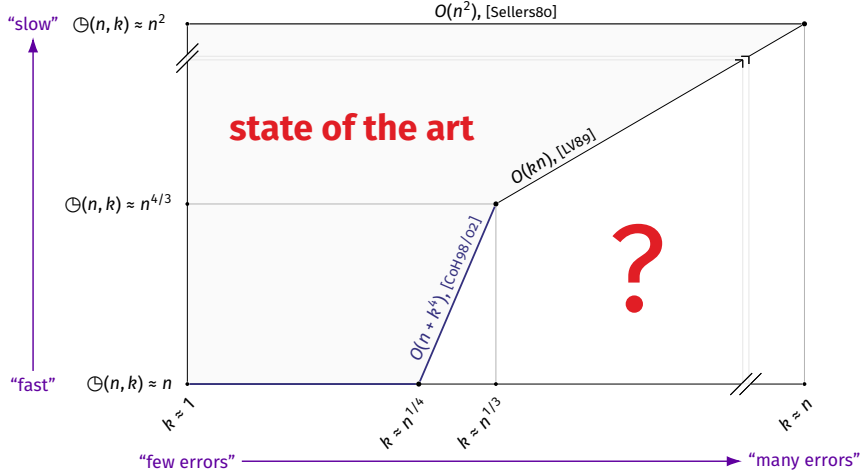
Existing algorithms for the case $n \approx t \approx p$

Recap: State-of-the-Art Algorithms



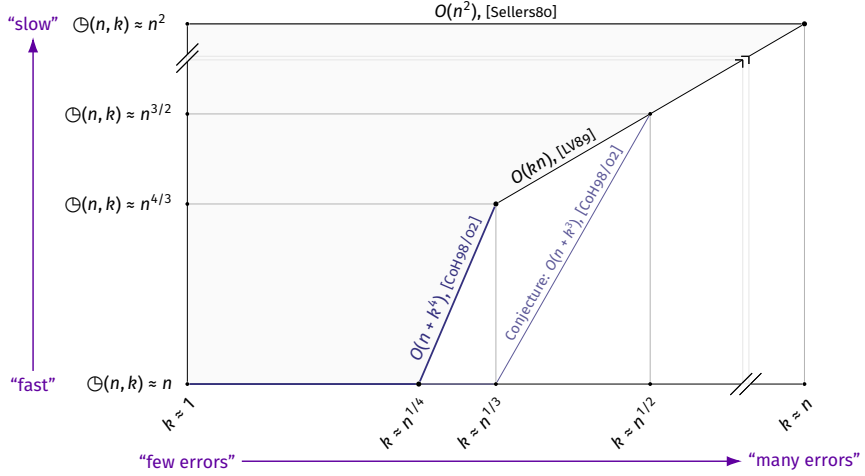
Existing algorithms for the case $n \approx t \approx p$

Recap: State-of-the-Art Algorithms



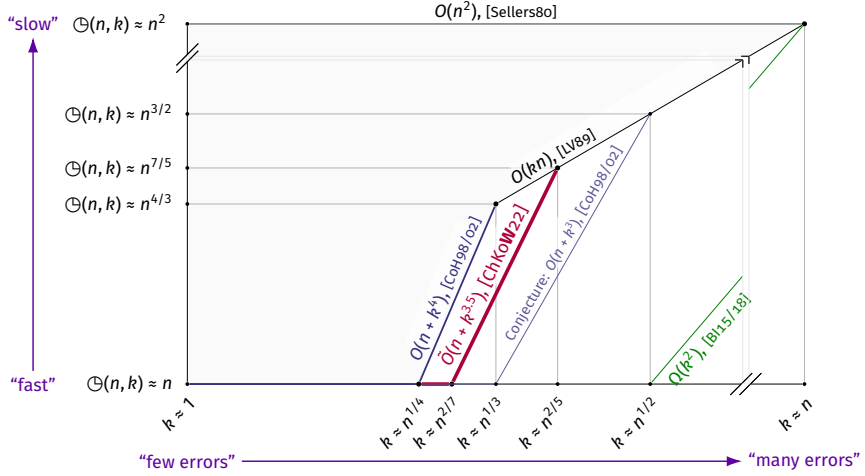
Existing algorithms for the case $n \approx t \approx p$

Recap: State-of-the-Art Algorithms



Existing algorithms for the case $n \approx t \approx p$

Recap: State-of-the-Art Algorithms



Existing algorithms for the case $n \approx t \approx p$

Why did a new improvement take 24 years?

Detour: Why did a new improvement for Approximate String Matching take 24 years?

- Simpler problems were not fully understood!

- Simpler problems were not fully understood!
- Example 1: The complexity of Edit Distance was settled only in 2014

Hardness of Edit Distance [BI18] (ann. 2014)

Computing the Edit Distance of to strings of length n is not possible in $O(n^{2-\varepsilon})$ time.

(Unless there is a major breakthrough for the Satisfiability Problem.)

- Simpler problems were not fully understood!
- Example 1: The complexity of Edit Distance was settled only in 2014

Hardness of Edit Distance [BI18] (ann. 2014)

Computing the Edit Distance of to strings of length n is not possible in $O(n^{2-\varepsilon})$ time.
(Unless there is a major breakthrough for the Satisfiability Problem.)

$\rightsquigarrow$ Yields lower bound of $O(t + k^2 \cdot t/p)$ for Approximate String Matching.

Detour: Why did a new improvement for Approximate String Matching take 24 years?

- Simpler problems were not fully understood!
- Example 2: String Matching with Mismatches (no insertions or deletions of chars)

- Simpler problems were not fully understood!
- Example 2: String Matching with Mismatches (no insertions or deletions of chars)
  - $\rightsquigarrow O(tk)$ algorithm [Landau, Vishkin'**86**]
  - $\rightsquigarrow \tilde{O}(t\sqrt{k})$ algorithm [Amir, Lewenstein, Porat'**04**]
  - $\rightsquigarrow \tilde{O}(t + t/p \cdot k^2)$ algorithm [CFPSS'**16**]

- Simpler problems were not fully understood!
- Example 2: String Matching with Mismatches (no insertions or deletions of chars)
  - $\rightsquigarrow O(tk)$ algorithm [Landau, Vishkin'**86**]
  - $\rightsquigarrow \tilde{O}(t\sqrt{k})$ algorithm [Amir, Lewenstein, Porat'**04**]
  - $\rightsquigarrow \tilde{O}(t + t/p \cdot k^2)$ algorithm [CFPSS'**16**]

[GU18] (ann. 2017)

Optimal String Matching with Mismatches

There is a $\tilde{O}(t + kt/\sqrt{p})$-time algorithm for String Matching with Mismatches and no significantly faster algorithm exits.

(Unless there is a major breakthrough for combinatorial Boolean Matrix Multiplication.)

- Simpler problems were not fully understood!
- Example 2: String Matching with Mismatches (no insertions or deletions of chars)
  - $\rightsquigarrow O(tk)$ algorithm [Landau, Vishkin'**86**]
  - $\rightsquigarrow \tilde{O}(t\sqrt{k})$ algorithm [Amir, Lewenstein, Porat'**04**]
  - $\rightsquigarrow \tilde{O}(t + t/p \cdot k^2)$ algorithm [CFPSS'**16**]

[GU18] (ann. 2017)

Optimal String Matching with Mismatches

There is a $\tilde{O}(t + kt/\sqrt{p})$-time algorithm for String Matching with Mismatches and no significantly faster algorithm exits.

(Unless there is a major breakthrough for combinatorial Boolean Matrix Multiplication.)

$\rightsquigarrow$ String Matching with Mismatches was "fully" understood only in 2017.

How do we obtain faster algorithms?

How do we obtain faster algorithms?

New insights into the solution structure of
Approximate String Matching

Think: Better filter + more structure when filter fails

Step 0:

What is the solution structure of
Exact String Matching?

(The) Period of a String

$p > 0$ is a period of a string S if $S[\,i\,] = S[\,i + p\,]$ for all $i = 1, \ldots, |S| - p$.

The period of S: smallest period of S, write $\mathrm{per}(S)$.

S is periodic: $\mathrm{per}(S) \leq |S|/2$

S = a b c a b c a b c a b

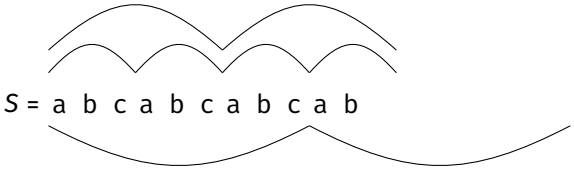
$\mathrm{per}(S) = 3$

(The) Period of a String

$p > 0$ is a period of a string S if $S[\,i\,] = S[\,i + p\,]$ for all $i = 1, \ldots, |S| - p$.

The period of S: smallest period of S, write per(S).

S is periodic: per(S) $\leq |S|/2$



per(S) = 3

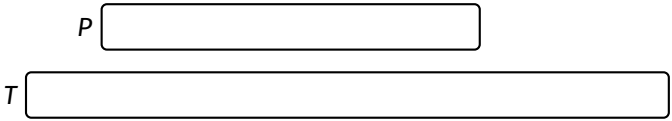
6 and 9 also periods of S

"Periodicity Lemma"

(Folklore)

Text T, pattern P with $t \leq \frac{3}{2}p$; one of the following holds

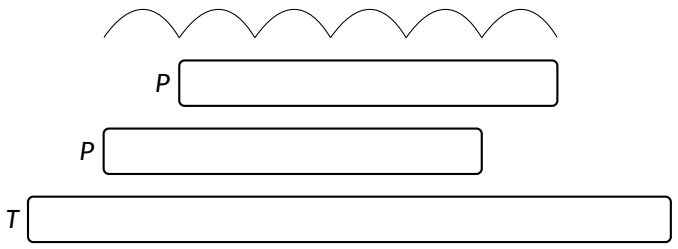
- P appears ≤ 1 times in T.



"Periodicity Lemma" (Folklore)

Text T, pattern P with $t \leq \frac{3}{2}p$; one of the following holds

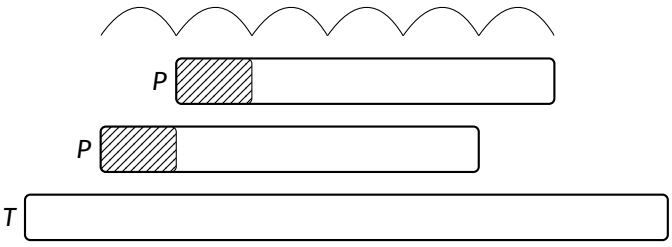
- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



"Periodicity Lemma" (Folklore)

Text T, pattern P with $t \leq \frac{3}{2}p$; one of the following holds

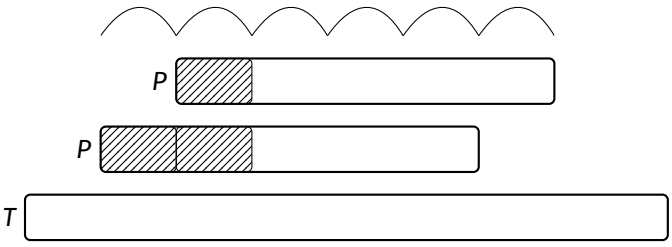
- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



"Periodicity Lemma" (Folklore)

Text T, pattern P with $t \leq \frac{3}{2}p$; one of the following holds

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.

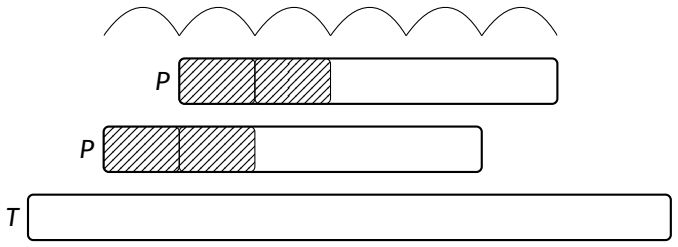


"Periodicity Lemma"

(Folklore)

Text T, pattern P with $t \leq \frac{3}{2}p$; one of the following holds

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.

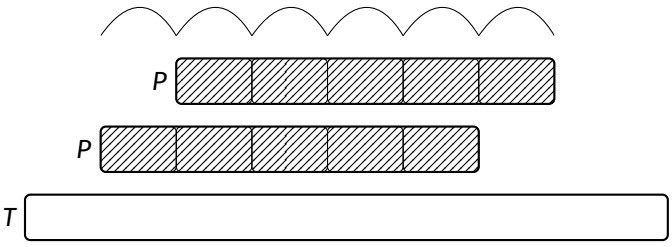


"Periodicity Lemma"

(Folklore)

Text T, pattern P with $t \leq \frac{3}{2}p$; one of the following holds

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.

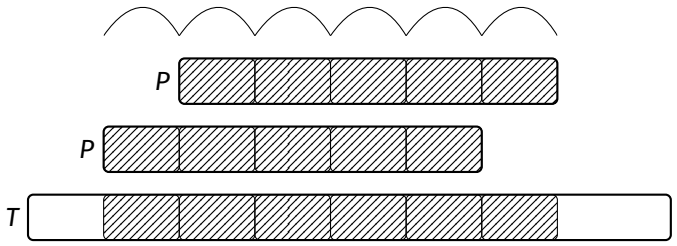


"Periodicity Lemma"

(Folklore)

Text T, pattern P with $t \leq \frac{3}{2}p$; one of the following holds

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.

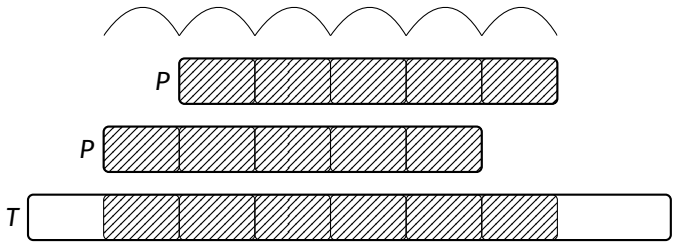


"Periodicity Lemma"

(Folklore)

Text T, pattern P with $t \leq \frac{3}{2}p$; one of the following holds

- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



Standard Trick is useful here:

For $t \gg p$ consider separately $O(t/p)$ fragments of T; then Periodicity Lemma applies

Step 1:

What is the solution structure of
String Matching with Mismatches?

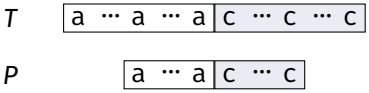
Periodicity Lemma (Folklore)

Text T, pattern P with $t \leq {}^3/_2\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches) [BKü**W**'19; ChKo**W**'20]

Text T, pattern P with $t \leq {}^3/_2\, p$; threshold k, one of the following holds

P appears $\leq O(k)$ times in T



a···ac···c appears $2k$ times

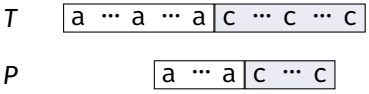
Periodicity Lemma (Folklore)

Text T, pattern P with $t \leq {}^3/_2\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches) [BKüW'19; ChKoW'20]

Text T, pattern P with $t \leq {}^3/_2\, p$; threshold k, one of the following holds

P appears $\leq O(k)$ times in T



a···ac···c appears $2k$ times

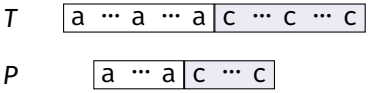
Periodicity Lemma (Folklore)

Text T, pattern P with $t \leq {}^3/_2\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

Main Result (Mismatches) [BKüW'19; ChKoW'20]

Text T, pattern P with $t \leq {}^3/_2\, p$; threshold k, one of the following holds

P appears $\leq O(k)$ times in T



a···ac···c appears $2k$ times

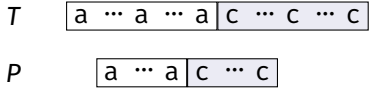
Periodicity Lemma (Folklore)

Text T, pattern P with $t \leq {}^3/_2\, p$; P appears ≤ 1 times in T or P is periodic with some period Q.

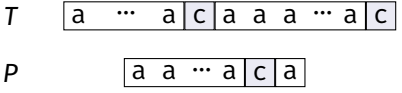
Main Result (Mismatches) [BKüW'19; ChKoW'20]

Text T, pattern P with $t \leq {}^3/_2\, p$; threshold k, one of the following holds

P appears $\leq O(k)$ times in T or P is almost periodic with some period Q



a···ac···c appears $2k$ times



aa···aca has approximate period a
(is at HD $\leq 2k$ from Q^∞)

Step 1.5:

The solution structure of
Approximate String Matching.

Main Result (Mismatches) [BKüW'19; ChKoW'20]

T, P with $t \leq 3/2\, p$; threshold k, P appears $O(k)$ times in T or P is almost periodic with some period Q

Main Result (Edits) [ChKoW'20]

Text T, pattern P with $t \leq 3/2\, p$; threshold k, one of the following holds

or P is almost periodic with some period Q

Main Result (Mismatches)

[BKüW'19; ChKoW'20]

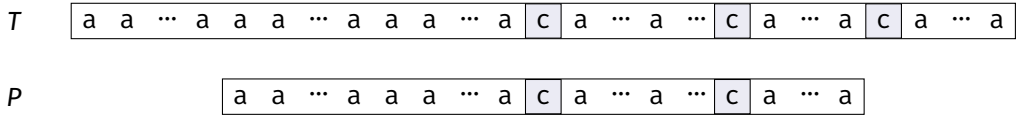
T, P with $t \leq {}^3/_2\, p$; threshold k, P appears $O(k)$ times in T or P is almost periodic with some period Q

Main Result (Edits)

[ChKoW'20]

Text T, pattern P with $t \leq {}^3/_2\, p$; threshold k, one of the following holds

P appears $\leq O(k^2)$ times in T or P is almost periodic with some period Q



aa···aaa···aca···a···ca···a appears $\approx k^2$ times

Main Result (Mismatches) [BKüW'19; ChKoW'20]

T, P with $t \leq {}^3/_2\, p$; threshold k, P appears $O(k)$ times in T or P is almost periodic with some period Q

Main Result (Edits) [ChKoW'20]

Text T, pattern P with $t \leq {}^3/_2\, p$; threshold k, one of the following holds

P appears $\leq O(k^2)$ times in T or P is almost periodic with some period Q

aa···aaa···aca···a···ca···a appears $\approx k^2$ times

Main Result (Mismatches) [BKüW'19; ChKoW'20]

T, P with $t \leq {}^3/_2\, p$; threshold k, P appears $O(k)$ times in T or P is almost periodic with some period Q

Main Result (Edits) [ChKoW'20]

Text T, pattern P with $t \leq {}^3/_2\, p$; threshold k, one of the following holds

P appears $\leq O(k^2)$ times in T or P is almost periodic with some period Q

aa···aaa···aca···a···ca···a appears $\approx k^2$ times

Main Result (Edits) [ChKoW'20]

T, P with $t \leq {}^3/_2\, p$; threshold k, P appears $O(k^2)$ times in T or P is almost periodic with some period Q

Main Result (Edits) [ChKoW'20]

T, P with $t \leq {}^{3}/_{2}\, p$; threshold k, P appears $O(k^2)$ times in T or P is almost periodic with some period Q

Key Intermediate Result (Analyze) [ChKoW'20]

Every string P satisfies at least one of

- P is almost periodic. aaacaaaaaaaaacaaaaaaa

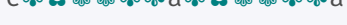
Main Result (Edits) [ChKoW'20]

T, P with $t \leq {}^{3}/_{2}\, p$; threshold k, P appears $O(k^2)$ times in T or P is almost periodic with some period Q

Key Intermediate Result (Analyze) [ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks



- P is almost periodic.

aaacaaaaaaaaacaaaaaaa

Main Result (Edits)

[ChKoW'20]

T, P with $t \leq {}^{3}/_{2}\, p$; threshold k, P appears $O(k^2)$ times in T or P is almost periodic with some period Q

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover ${}^{3}/_{8}\, P$ — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa

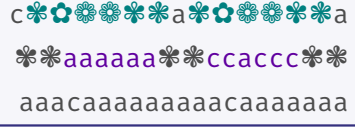
Main Result (Edits) [ChKoW'20]

T, P with $t \le {}^3/_2\, p$; threshold k, P appears $O(k^2)$ times in T or P is almost periodic with some period Q

Key Intermediate Result (Analyze) [ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks
- P has disjoint repetitive regions that cover ${}^3/_8\, P$
- P is almost periodic.



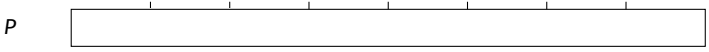
Analyze implies Main Result:
Almost periodicity $\rightsquigarrow$ as in Main Result
Breaks, repetitive regions $\rightsquigarrow$ good filter (as in [Cole, Hariharan'98])

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa

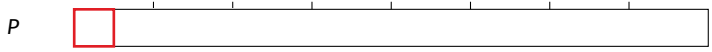


Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



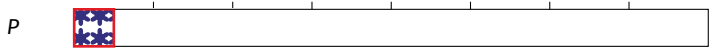
- Process P from left to right, $p/8k$ new characters at a time.

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❃✿❁❁❃❃a❃✿❁❁❃❃a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❃❃aaaaaa❃❃ccaccc❃❃
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



Breaks B = {  }

Repetitive Regions R = { }

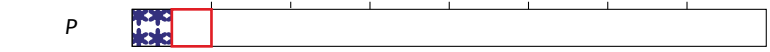
- If a fragment is a break, add it to the found breaks.

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



Breaks B = {  }

Repetitive Regions R = { }

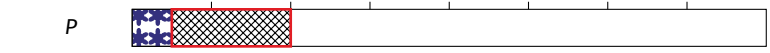
- Otherwise, find the shortest prefix (longer than $p/8k$) that is a repetitive region.

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



Breaks B = {  }

Repetitive Regions R = {  }

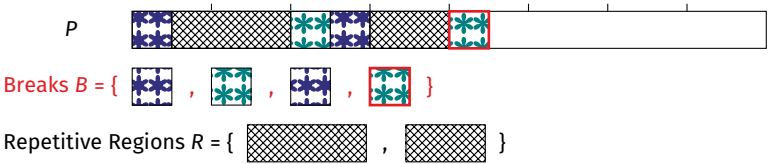
- Otherwise, find the shortest prefix (longer than $p/8k$) that is a repetitive region.

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



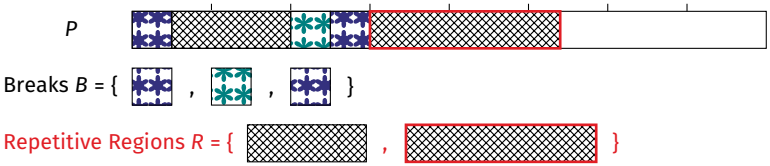
- If we found $2k$ breaks, return the breaks.

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



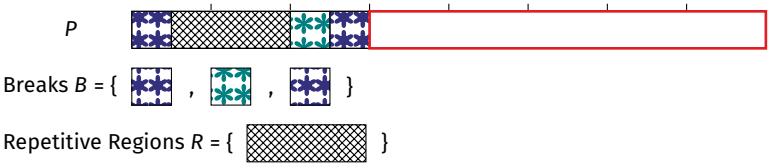
- If the total length of the repetitive regions is $> 3/8 \cdot p$, return the repetitive regions.

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c✾✿❁❁✾✾a✾✿❁❁✾✾a
- P has disjoint repetitive regions that cover $^3/_8$ P — ✾✾aaaaaa✾✾ccaccc✾✾
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



- If we reach the end of P, try to find a single repetitive region starting from the end.

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa

Breaks B = { }

Repetitive Regions R = { }

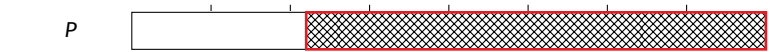
- If we reach the end of P, try to find a single repetitive region starting from the end.

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa



Breaks B = { }

Repetitive Regions R = { }

- If we found a repetitive region, return it.

Key Intermediate Result (Analyze)

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa

Breaks B = { }

Repetitive Regions R = { }

- If we again don't obtain a repetitive region, P is almost periodic.

Key Intermediate Result (Analyze) ✓

[ChKoW'20]

Every string P satisfies at least one of

- P has $2k$ disjoint, long breaks — c❀✿❁❁❀❀a❀✿❁❁❀❀a
- P has disjoint repetitive regions that cover $^3/_8$ P — ❀❀aaaaaa❀❀ccaccc❀❀
- P is almost periodic. — aaacaaaaaaaaacaaaaaaa

How do we turn our insights
into faster algorithms?

How do we turn our insights into faster algorithms?

Need to tackle three cases.

- P contains $2k$ disjoint breaks;
- P contains disjoint repetitive regions R_i;
- P is almost periodic

How do we turn our insights into faster algorithms?

Need to tackle ~~three~~ two cases.

- P contains $2k$ disjoint breaks;
- ~~P contains disjoint repetitive regions R_i;~~ $\rightsquigarrow$ Follows from the other two cases.
- P is almost periodic

How do we turn our insights into faster algorithms?

Need to tackle ~~three~~ one case.

- ~~P contains $2k$ disjoint breaks;~~ $\rightsquigarrow$ Adaption of [Cole,Hariharan'98] yields $O(|T| + |T|/|P| \cdot k^3)$.
- ~~P contains disjoint repetitive regions R_i;~~ $\rightsquigarrow$ Follows from the other two cases.
- P is almost periodic

How do we turn our insights into faster algorithms?

Need to tackle ~~three~~ one case.

- ~~P contains $2k$ disjoint breaks;~~ $\rightsquigarrow$ Adaption of [Cole,Hariharan'98] yields $O(|T| + |T|/|P| \cdot k^3)$.
- ~~P contains disjoint repetitive regions R_i;~~ $\rightsquigarrow$ Follows from the other two cases.
- P is almost periodic

[ChKo**W**'22]

Reduction to Periodic Patterns

Algorithm for almost periodic case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$, for $a \geq 3$
$\Longrightarrow$ Algorithm for general case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$

[ChKo**W**'22]

Reduction to Periodic Patterns

Algorithm for almost periodic case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$, for $a \geq 3$

$\Longrightarrow$ Algorithm for general case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$

Need to tackle: P is almost periodic

Reduction to Periodic Patterns [ChKo**W**'22]

Algorithm for almost periodic case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$, for $a \geq 3$
$\Longrightarrow$ Algorithm for general case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$

Need to tackle: P is almost periodic

- In [ChKo**W**'20], use elaborate marking scheme to obtain $O(|T| + |T|/|P| \cdot k^4)$ algorithm
  $\rightsquigarrow$ Not faster than [Cole,Hariharan'98]

[ChKo**W**'22]

Reduction to Periodic Patterns

Algorithm for almost periodic case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$, for $a \geq 3$
$\Longrightarrow$ Algorithm for general case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$

Need to tackle: P is almost periodic

- In [ChKo**W**'20], use elaborate marking scheme to obtain $O(|T| + |T|/|P| \cdot k^4)$ algorithm
  $\rightsquigarrow$ Not faster than [Cole,Hariharan'98]
- In [ChKo**W**'22], trade-off between
  - Refinement of algorithm from [ChKo**W**'20]
  - New algorithm based on "Seaweed Technology" of [Tiskin'10,'15]

  $\rightsquigarrow$ Yields $O(|T| + |T|/|P| \cdot k^{3.5})$ algorithm

Reduction to Periodic Patterns

[ChKo**W**'22]

Algorithm for almost periodic case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$, for $a \geq 3$

$\Longrightarrow$ Algorithm for general case in time $\tilde{O}(|T| + k^a \cdot |T|/|P|)$

Need to tackle: P is almost periodic

- In [ChKo**W**'20], use elaborate marking scheme to obtain $O(|T| + |T|/|P| \cdot k^4)$ algorithm

  $\rightsquigarrow$ Not faster than [Cole,Hariharan'98]
- In [ChKo**W**'22], trade-off between
  - Refinement of algorithm from [ChKo**W**'20]
  - ~~New algorithm based on "Seaweed Technology" of [Tiskin'10,'15]~~
    New in [ChKo**W**'25]: Simpler plug-in replacement for "Seaweed Technology" based on SMAWK (efficient $(\min, +)$-multiplication of Monge matrices)

  $\rightsquigarrow$ Yields $O(|T| + |T|/|P| \cdot k^{3.5})$ algorithm

Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - $\texttt{LCP}(S, T)$: Compute the length of the longest common prefix of S and T.
  - $\texttt{LCP}^R(S, T)$: Compute the length of the longest common suffix of S and T.
  - $\texttt{IPM}(P, T)$: Compute all exact matches of P in T.
  - $\texttt{Length}(S)$: Compute the length s of S.
  - $\texttt{Access}(S, i)$: Retrieve the character $S[i]$.
  - $\texttt{Extract}(S, \ell, r)$: Extract the fragment (or substring) $S[\ell..r]$ from S.

Main Result (PILLAR Algorithm) [ChKoW'22]

For $t \leq 3/2\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations

Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP$^R(S, T)$: Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - ExtRact(S, ℓ, r): Extract the fragment (or substring) $S[\ell..r)$ from S.

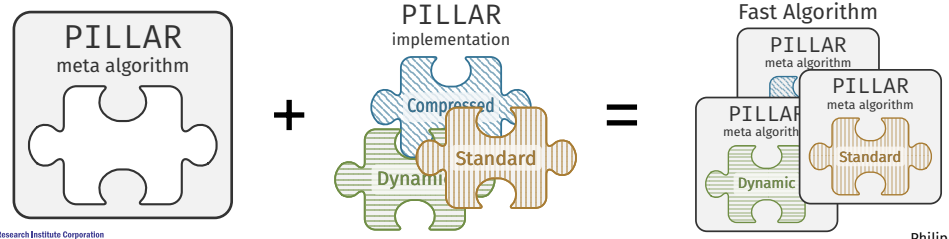
Main Result (PILLAR Algorithm) [ChKoW'22]

For $t \leq 3/2\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP$^R(S, T)$: Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - ExtRact(S, ℓ, r): Extract the fragment (or substring) $S[\ell..r]$ from S.

Main Result (PILLAR Algorithm) [ChKoW'22]

For $t \le 3/2\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations



Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP$^R(S, T)$: Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - ExtRact(S, ℓ, r): Extract the fragment (or substring) $S[\ell..r)$ from S.

Main Result (PILLAR Algorithm) [ChKoW'22]

For $t \leq 3/2\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations

- In [ChKoW'20]: efficient PILLAR implementations for
  - Standard setting (each op $O(1)$ after global $O(t)$ preprocessing)
  - Fully-compressed setting (T, P grammar-compressed to size n; each op $\tilde{O}(1)$ after $\tilde{O}(n)$ prep.)
    (In this setting, already [ChKoW'20] obtained faster algorithms)
  - Dynamic Setting

Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP$^R(S, T)$: Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - ExtRact(S, ℓ, r): Extract the fragment (or substring) $S[\ell..r)$ from S.

Main Result (PILLAR Algorithm) [ChKoW'22]

For $t \leq 3/2\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations

- In [ChKoW'20]: efficient PILLAR implementations for
  - Standard setting (each op $O(1)$ after global $O(t)$ preprocessing)
  - Fully-compressed setting (T, P grammar-compressed to size n; each op $\tilde{O}(1)$ after $\tilde{O}(n)$ prep.)
    (In this setting, already [ChKoW'20] obtained faster algorithms)
  - Dynamic Setting
- Since then: quantum setting (more details soon!), packed setting, read-only setting, ...
  PILLAR algorithms for other related problems, etc. [ChKoW'20] has > 60 citations by now

Beyond Strings: The PILLAR Model

- All of our algorithms rely on a small set of essential operations:
  - LCP(S, T): Compute the length of the longest common prefix of S and T.
  - LCP$^R(S, T)$: Compute the length of the longest common suffix of S and T.
  - IPM(P, T): Compute all exact matches of P in T.
  - Length(S): Compute the length s of S.
  - Access(S, i): Retrieve the character $S[i]$.
  - ExtRact(S, ℓ, r): Extract the fragment (or substring) $S[\ell..r)$ from S.

Main Result (PILLAR Algorithm) [ChKoW'22]

For $t \le 3/2\, p + k$; ASM can be solved using $O(k^{3.5})$ PILLAR operations

- In [ChKoW'20]: efficient PILLAR implementations for
  - Standard setting (each op $O(1)$ after global $O(t)$ preprocessing)
  - Fully-compressed setting (T, P grammar-compressed to size n; each op $\tilde{O}(1)$ after $\tilde{O}(n)$ prep.)
    (In this setting, already [ChKoW'20] obtained faster algorithms)
  - Dynamic Setting
- Since then: quantum setting (more details soon!), packed setting, read-only setting, ...
  PILLAR algorithms for other related problems, etc. [ChKoW'20] has > 60 citations by now
- Bottom-line, useful design principle: meta algorithm (PILLAR) + simple-to-solve subproblems

Spotlight Extension: Quantum Algorithms for ASM

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Query complexity $Q(n)$: number of oracle queries
- Time complexity $T(n)$: number of elementary gates

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Query complexity $Q(n)$: number of oracle queries
- Time complexity $T(n)$: number of elementary gates
- Goal: algorithms with $Q(n) \ll n$ and $T(n) \ll n$

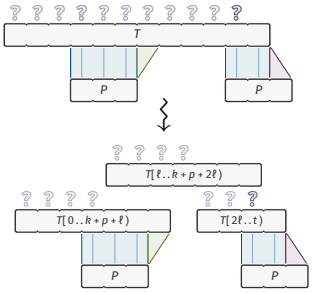
Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Query complexity $Q(n)$: number of oracle queries
- Time complexity $T(n)$: number of elementary gates
- Goal: algorithms with $Q(n) \ll n$ and $T(n) \ll n$

Toy example: "Standard Trick" for ASM decision version (∃? occ)
Split T into overlapping fragments of len $\ell + p + k$
$\rightsquigarrow O(t/\ell)$ instances, each "responsible" for its first ℓ positions
$\rightsquigarrow$ Classically: $O(t/\ell)$ time overhead (evaluate each inst. 1 by 1)



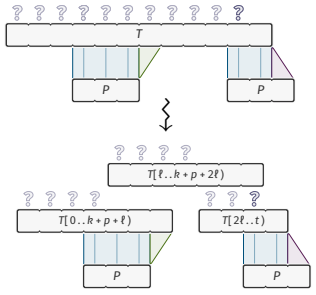
Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Query complexity $Q(n)$: number of oracle queries
- Time complexity $T(n)$: number of elementary gates
- Goal: algorithms with $Q(n) \ll n$ and $T(n) \ll n$

Toy example: "Standard Trick" for ASM decision version (∃? occ)
Split T into overlapping fragments of len $\ell + p + k$
$\rightsquigarrow O(t/\ell)$ instances, each "responsible" for its first ℓ positions
$\rightsquigarrow$ Classically: $O(t/\ell)$ time overhead (evaluate each inst. 1 by 1)
Quantum setting: Use amplitude amp. / Grover's algo.
$\tilde{O}(\sqrt{t/\ell})$ time and $O(\sqrt{t/\ell})$ evaluations of single inst.



Amplitude Amplification

[Gro96, BHMT02]

Given function $f : [0..n) \to \{0, 1\}$
Can (wp ≥ 2/3) obtain $x \in f^{-1}(1)$ in $\tilde{O}(\sqrt{n})$ time + $O(\sqrt{n})$ queries to f

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Idea: Use existing quantum algorithms to implement PILLAR operations.

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Idea: Use existing quantum algorithms to implement PILLAR operations.

`LCP(S,T)`; `IPM(P,T)`; `Length(S)`; `Access(S,i)`; `ExtRact(S,ℓ,r)`

PILLAR Algorithm

[ChKoW'22]

For $t \leq 3/2\, p + k$;
ASM can be solved
using $O(k^{3.5})$ PILLAR operations

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Idea: Use existing quantum algorithms to implement PILLAR operations.

LCP(S,T); IPM(P,T); Length(S); Access(S,i); ExtRact(S,ℓ,r)

PILLAR Algorithm [ChKoW'22]

For $t \leq 3/2\, p + k$;
ASM can be solved
using $O(k^{3.5})$ PILLAR operations

Quantum PILLAR Impl implied by [HV'03]

For length-n strings,
can implement all PILLAR ops
in $\tilde{O}(\sqrt{n})$ time and queries

Yields $\tilde{O}(t/p \cdot k^{3.5} \cdot \sqrt{p})$ quantum time algo for ASM; ($\tilde{O}(\sqrt{t/p} \cdot k^{3.5} \cdot \sqrt{p})$ for $\exists$?) (algos are correct whp.)

early 1980's

Approximate String Matching

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Idea: Use existing quantum algorithms to implement PILLAR operations.

`LCP(S,T); IPM(P,T); Length(S); Access(S,i); ExtRact(S,ℓ,r)`

PILLAR Algorithm [ChKoW'22]

For $t \leq 3/2\, p + k$;
ASM can be solved
using $O(k^{3.5})$ PILLAR operations

Quantum PILLAR Impl implied by [HV'03]

For length-n strings,
can implement all PILLAR ops
in $\tilde{O}(\sqrt{n})$ time and queries

Yields $\tilde{O}(t/p \cdot k^{3.5} \cdot \sqrt{p})$ quantum time algo for ASM; ($\tilde{O}(\sqrt{t/p} \cdot k^{3.5} \cdot \sqrt{p})$ for $\exists$?) (algos are correct whp.)

Are these algorithms optimal? $\rightsquigarrow$ No.

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T

$\rightsquigarrow$ k-edit-aware lossy compression of P and T (think: filter + discarding filtered-out parts)

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T
$\rightsquigarrow$ k-edit-aware lossy compression of P and T (think: filter + discarding filtered-out parts)

In [KoN**W**'24, KoN**W**'25] for T and P with $t \leq {}^3\!/_2\, p + k$:

- Show: any k-edit-aware lossy compr. of P and T needs $\Omega(k \log(t/k))$ bits

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T
$\rightsquigarrow$ k-edit-aware lossy compression of P and T (think: filter + discarding filtered-out parts)

In [KoN**W**'24, KoN**W**'25] for T and P with $t \leq {}^3\!/_2\, p + k$:

- Show: any k-edit-aware lossy compr. of P and T needs $\Omega(k \log(t/k))$ bits
- Obtain k-edit-aware lossy compr. of P and T as $\tilde{O}(k)$ substring equations; computable via quantum algorithm using $\hat{O}(\sqrt{kp})$ queries/time

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T
$\rightsquigarrow$ k-edit-aware lossy compression of P and T (think: filter + discarding filtered-out parts)

In [KoN**W**'24, KoN**W**'25] for T and P with $t \leq {}^3\!/_2\, p + k$:

- Show: any k-edit-aware lossy compr. of P and T needs $\Omega(k \log(t/k))$ bits
- Obtain k-edit-aware lossy compr. of P and T as $\tilde{O}(k)$ substring equations;
  computable via quantum algorithm using $\hat{O}(\sqrt{kp})$ queries/time
- Give classical $\tilde{O}(b^2)$-time algo to compute sol. to b substr. eqn's as $O(b)$-size grammar

Approximate String Matching

early 1980's

Given: text T, pattern P, integer k; Find: (starting pos. of) substrings of T at edit distance $\leq k$ to P.

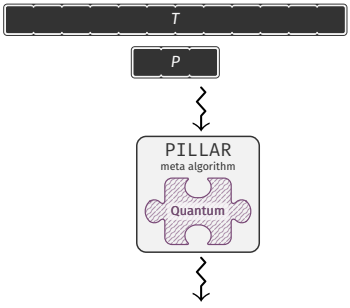
- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

Challenge: query as few char's as possible, but enough to recover all k-edit occ's of P in T
$\rightsquigarrow$ k-edit-aware lossy compression of P and T (think: filter + discarding filtered-out parts)

In [KoN**W**'24, KoN**W**'25] for T and P with $t \leq {}^3\!/_2\, p + k$:

- Show: any k-edit-aware lossy compr. of P and T needs $\Omega(k \log(t/k))$ bits
- Obtain k-edit-aware lossy compr. of P and T as $\tilde{O}(k)$ substring equations;
  computable via quantum algorithm using $\hat{O}(\sqrt{kp})$ queries/time
- Give classical $\tilde{O}(b^2)$-time algo to compute sol. to b substr. eqn's as $O(b)$-size grammar
  $\rightsquigarrow$ Can use `PILLAR` algorithm for fully-compressed setting on lossy compression!

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

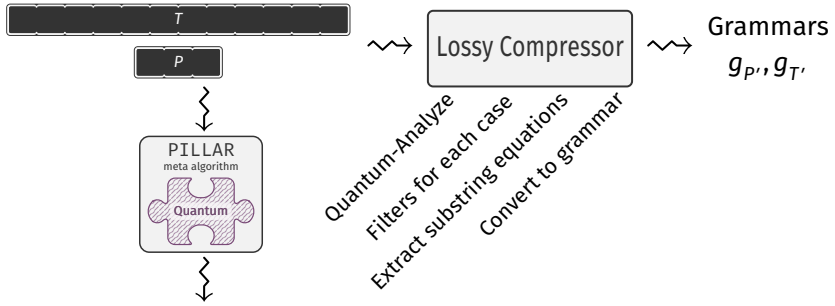


Easy Result

ASM on T, P w/ $t \leq {}^{3}/_{2}\, p + k$ admits qu-algo using $\tilde{O}(k^{3.5}\sqrt{n})$ queries/time

Quantum ASM: Bottom-Lines

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

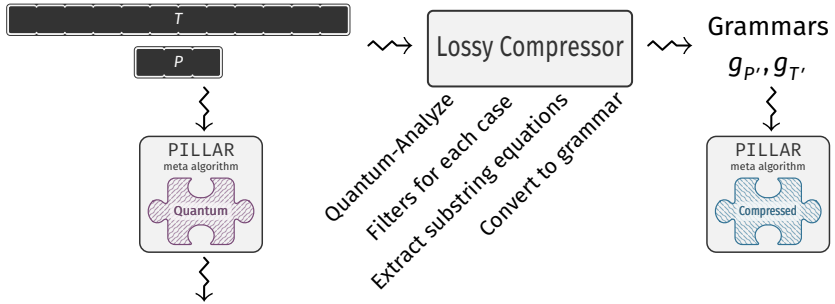


Easy Result

ASM on T, P w/ $t \leq 3/2\, p + k$ admits qu-algo using $\tilde{O}(k^{3.5}\sqrt{n})$ queries/time

Quantum ASM: Bottom-Lines

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$

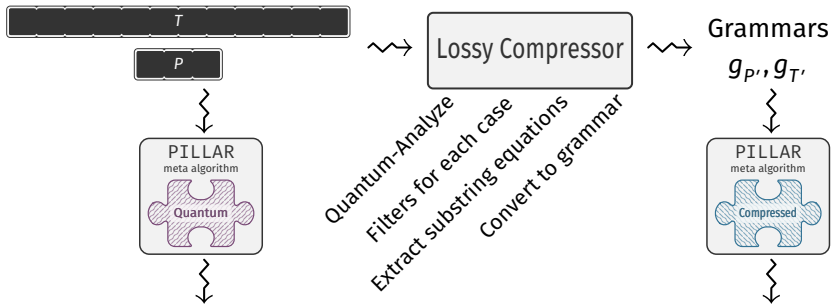


Easy Result

ASM on T, P w/ $t \leq {}^3\!/_2\, p + k$ admits qu-algo using $\tilde{O}(k^{3.5}\sqrt{n})$ queries/time

Quantum ASM: Bottom-Lines

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$



Easy Result

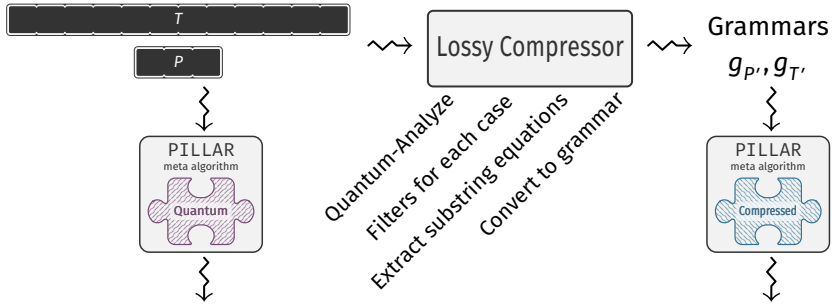
ASM on T, P w/ $t \le 3/2\, p + k$ admits qu-algo using $\tilde{O}(k^{3.5}\sqrt{n})$ queries/time

Better Result [KoNW'24,'25]

ASM on T, P w/ $t \le 3/2\, p + k$ admits qu-algo using $\hat{O}(\sqrt{kn})$ queries + $\hat{O}(\sqrt{kn} + k^{3.5})$ time

Quantum ASM: Bottom-Lines

- P/T given as oracle $\rightsquigarrow$ queries possible in superposition
- Goal: algorithms with query complexity $Q(n) \ll n$ and time complexity $T(n) \ll n$



Easy Result

ASM on T, P w/ $t \le 3/2\, p + k$ admits qu-algo using $\tilde{O}(k^{3.5}\sqrt{n})$ queries/time

Better Result [KoNW'24,'25]

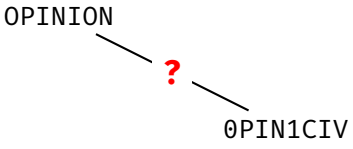
ASM on T, P w/ $t \le 3/2\, p + k$ admits qu-algo using $\hat{O}(\sqrt{kn})$ queries + $\hat{O}(\sqrt{kn} + k^{3.5})$ time

$\rightsquigarrow$ Queries are (near) optimal [KoN**W**'24], time overhead from PILLAR (but future-proof)

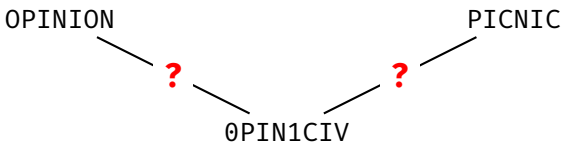
Spotlight Extension: String Matching with Weighted Edits

How similar are two strings X and Y?

How similar are two strings X and Y?



How similar are two strings X and Y?

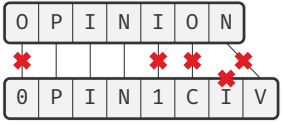


Edit Distance $\mathrm{ED}(X, Y)$

Min number of character insertions, deletions, and substitutions that transform X to Y

Edit Distance $\mathrm{ED}(X, Y)$

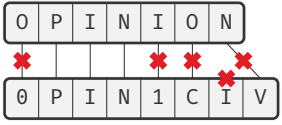
Min number of character insertions, deletions, and substitutions that transform X to Y



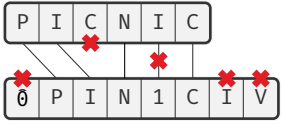
$\mathrm{ED}(\texttt{0PIN1CIV}, \texttt{OPINION}) = 5$

Edit Distance $\mathrm{ED}(X, Y)$

Min number of character insertions, deletions, and substitutions that transform X to Y



$\mathrm{ED}(\texttt{0PIN1CIV}, \texttt{OPINION}) = 5$



$\mathrm{ED}(\texttt{0PIN1CIV}, \texttt{PICNIC}) = 5$

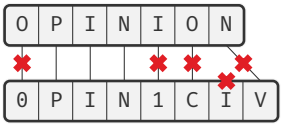
Weighted Edit Distance

$\mathrm{ED}^w(X, Y)$

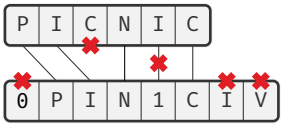
Min cost of transforming X to Y using character edits, where:

- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(0, 0) := 1 \quad w(1, \mathrm{I}) := 1 \quad w(\mathrm{C}, 0) := 1$



$w(*, *) := 2 \quad w(*, \varepsilon) := 1 \quad w(\varepsilon, *) := 10$



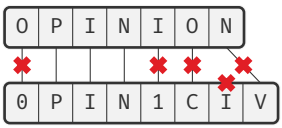
Weighted Edit Distance

$\mathrm{ED}^w(X, Y)$

Min cost of transforming X to Y using character edits, where:

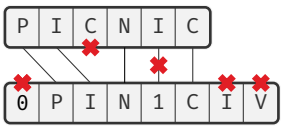
- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(0, \mathtt{O}) := 1 \quad w(1, \mathtt{I}) := 1 \quad w(\mathtt{C}, \mathtt{O}) := 1$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{OPINION}) = 6$

$w(*, *) := 2 \quad w(*, \varepsilon) := 1 \quad w(\varepsilon, *) := 10$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{PICNIC}) \leq 14$

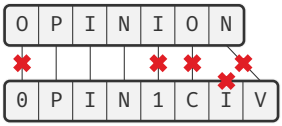
$\mathrm{ED}^w(X, Y)$

Weighted Edit Distance

Min cost of transforming X to Y using character edits, where:

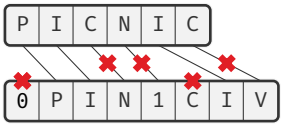
- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(0, \mathtt{O}) := 1 \quad w(1, \mathtt{I}) := 1 \quad w(\mathtt{C}, \mathtt{O}) := 1$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{OPINION}) = 6$

$w(*, *) := 2 \quad w(*, \varepsilon) := 1 \quad w(\varepsilon, *) := 10$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{PICNIC}) = 8$

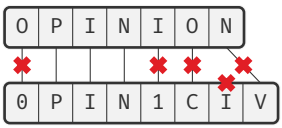
Weighted Edit Distance

$\mathrm{ED}^w(X, Y)$

Min cost of transforming X to Y using character edits, where:

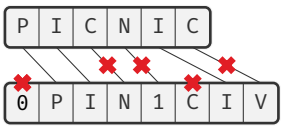
- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(0, O) := 1 \quad w(1, I) := 1 \quad w(C, 0) := 1$



$\mathrm{ED}^w(0\mathrm{PIN1CIV}, \mathrm{OPINION}) = 6$

$w(*, *) := 2 \quad w(*, \varepsilon) := 1 \quad w(\varepsilon, *) := 10$



$\mathrm{ED}^w(0\mathrm{PIN1CIV}, \mathrm{PICNIC}) = 8$

Justified Assumption: w is normalized, $w(x, y) \geq 1$ for all $x \neq y$.

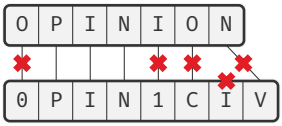
$\mathrm{ED}^w(X, Y)$

Weighted Edit Distance

Min cost of transforming X to Y using character edits, where:

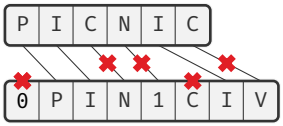
- inserting y costs $w(\varepsilon, y)$;
- deleting x costs $w(x, \varepsilon)$;
- substituting x for y costs $w(x, y)$.

$w(0, \mathtt{O}) := 1 \quad w(1, \mathtt{I}) := 1 \quad w(\mathtt{C}, 0) := 1$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{OPINION}) = 6$

$w(*, *) := 2 \quad w(*, \varepsilon) := 1 \quad w(\varepsilon, *) := 10$



$\mathrm{ED}^w(\mathtt{0PIN1CIV}, \mathtt{PICNIC}) = 8$

Justified Assumption: w is normalized, $w(x, y) \geq 1$ for all $x \neq y$.

Otherwise: could scale weights and condition $\mathrm{ED}^w_{\leq k}(X, Y) \leq k$ (e.g. in ASM) becomes meaningless.

Recall crucial subroutine for ASM:

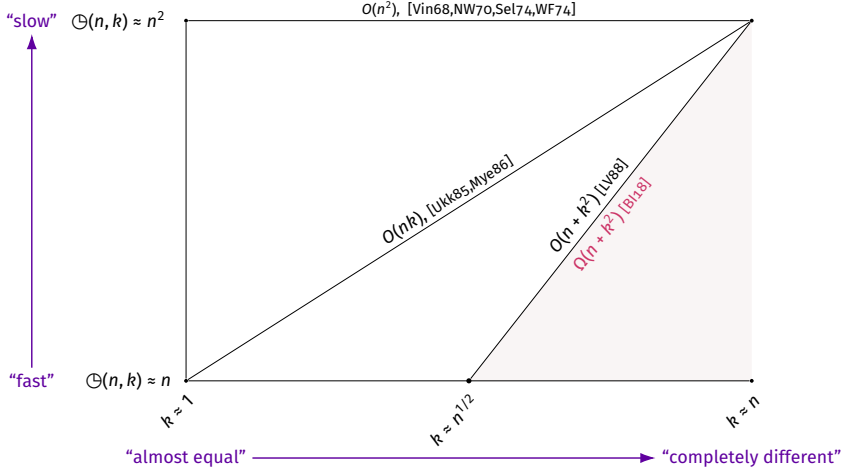
Verify that occurrence starts at given (interval of) position of T

Recall crucial subroutine for ASM:

Verify that occurrence starts at given (interval of) position of T

$\rightsquigarrow$ Need to compute Bounded Weighted Edit Distance

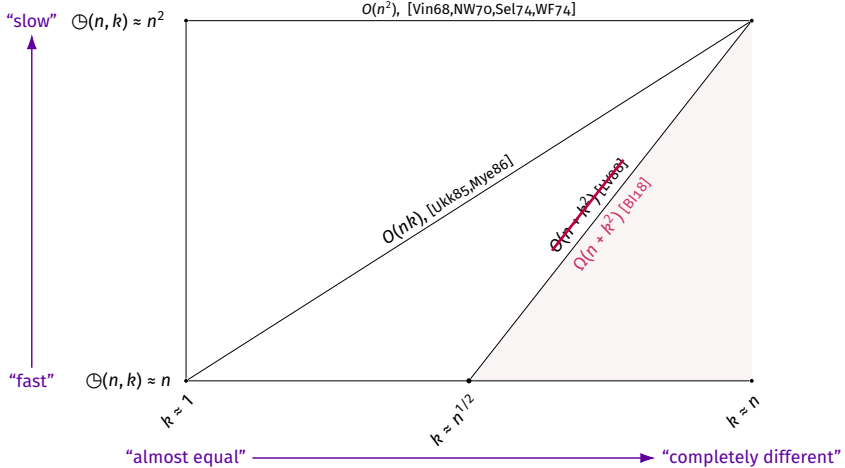
Recall: Algorithms for (Bounded) Edit Distance / Verify



Existing algorithms for Edit Distance ED(X, Y), where $|X|, |Y| \leq n$

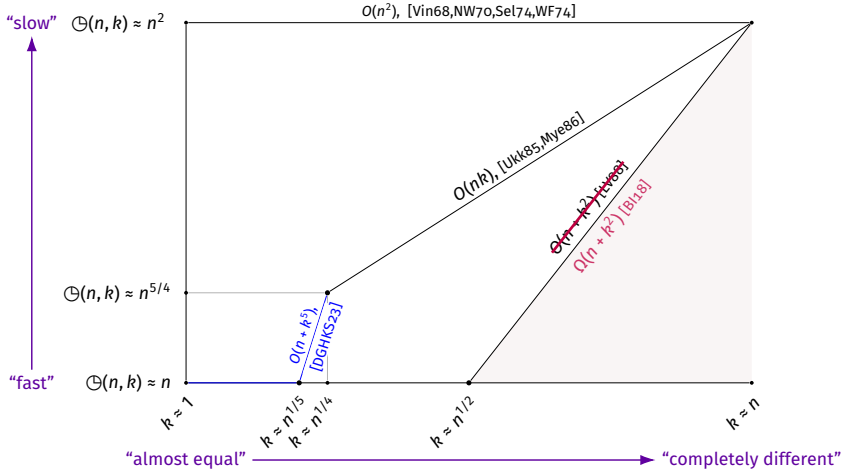
What about Weighted Edit Distance?

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify



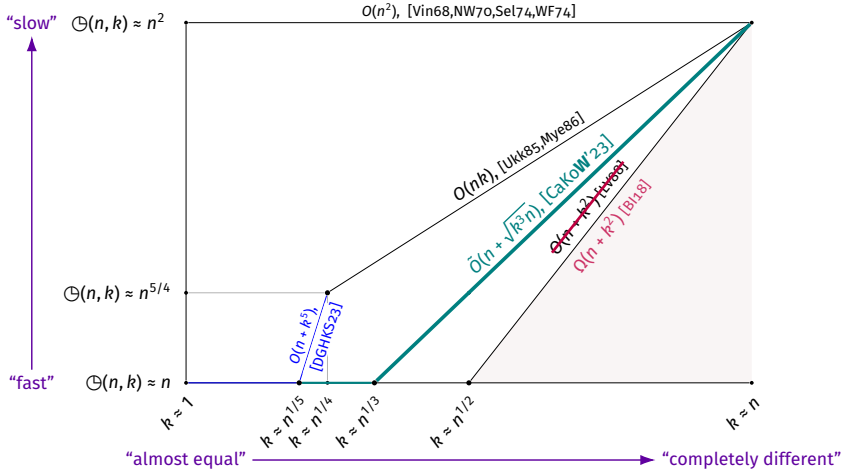
Existing algorithms for Weighted Edit Distance $\mathrm{ED}^w(X, Y)$, where $|X|, |Y| \leq n$

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify



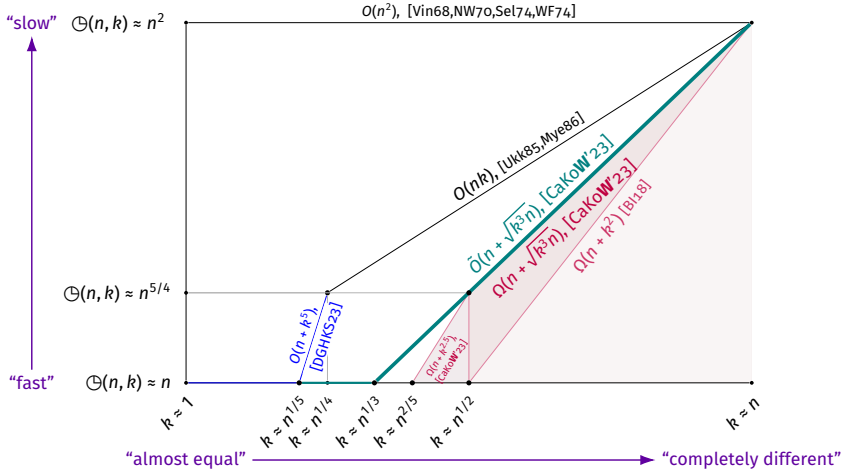
Existing algorithms for Weighted Edit Distance $\mathrm{ED}^w(X, Y)$, where $|X|, |Y| \leq n$

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify



Existing algorithms for Weighted Edit Distance $\mathrm{ED}^w(X, Y)$, where $|X|, |Y| \le n$

Algorithms for (Bounded) Weighted Edit Distance / Weighted Verify



Existing algorithms for Weighted Edit Distance $\mathrm{ED}^w(X, Y)$, where $|X|, |Y| \le n$

Main Theorem (Upper Bound) [CaKoW'23]

Strings X, Y each of length at most n

Oracle access to (normalized) weight function w

Can compute $k = \mathrm{ED}^w(X, Y)$ in time $O(n + \sqrt{nk^3} \log^3 n)$

Main Theorem (Upper Bound) [CaKoW'23]

Strings X, Y each of length at most n
Oracle access to (normalized) weight function w
Can compute $k = \mathrm{ED}^w(X, Y)$ in time $O(n + \sqrt{nk^3} \log^3 n)$

Main Theorem (Lower Bound) [CaKoW'23]

Assuming the APSP Hypothesis and for $\sqrt{n} \le k \le n$,
Main Theorem (Upper Bound) is tight (up to $n^{o(1)}$-factors)

Main Theorem (Upper Bound) [CaKoW'23]

Strings X, Y each of length at most n
Oracle access to (normalized) weight function w
Can compute $k = \mathrm{ED}^w(X, Y)$ in time $O(n + \sqrt{nk^3} \log^3 n)$

Main Theorem (Lower Bound) [CaKoW'23]

Assuming the APSP Hypothesis and for $\sqrt{n} \le k \le n$,
Main Theorem (Upper Bound) is tight (up to $n^{o(1)}$-factors)

Can extend UB to verify $\le k$ consecutive positions

Main Theorem (Upper Bound) [CaKo**W**'23]

Strings X, Y each of length at most n
Oracle access to (normalized) weight function w
Can compute $k = \mathrm{ED}^w(X, Y)$ in time $O(n + \sqrt{nk^3} \log^3 n)$

Main Theorem (Lower Bound) [CaKo**W**'23]

Assuming the APSP Hypothesis and for $\sqrt{n} \le k \le n$,
Main Theorem (Upper Bound) is tight (up to $n^{o(1)}$-factors)

Can extend UB to verify $\le k$ consecutive positions
$\rightsquigarrow$ Good enough for $\tilde{O}(t + t/p \cdot k^4)$ algo for weighted ASM (long, boring) [ChKo**W**'25]

Main Theorem (Upper Bound) [CaKoW'23]

Strings X, Y each of length at most n
Oracle access to (normalized) weight function w
Can compute $k = \mathrm{ED}^w(X, Y)$ in time $O(n + \sqrt{nk^3}\log^3 n)$

Main Theorem (Lower Bound) [CaKoW'23]

Assuming the APSP Hypothesis and for $\sqrt{n} \le k \le n$,
Main Theorem (Upper Bound) is tight (up to $n^{o(1)}$-factors)

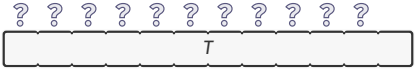
Can extend UB to verify $\le k$ consecutive positions
$\rightsquigarrow$ Good enough for $\tilde{O}(t + t/p \cdot k^4)$ algo for weighted ASM (long, boring) [ChKoW'25]

$\rightsquigarrow$ What about $\tilde{O}(kt)$-time for weighted ASM?
With Standard Trick, UB yields $\tilde{O}(t/k \cdot \sqrt{pk^3}) = \tilde{O}(t\sqrt{pk})$
LB rules out $\tilde{O}(kt)$ via better WED algo $\rightsquigarrow$ need different approach

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Focus: Obtain starting positions of occurrences

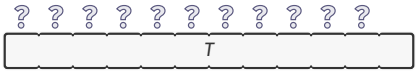


Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)



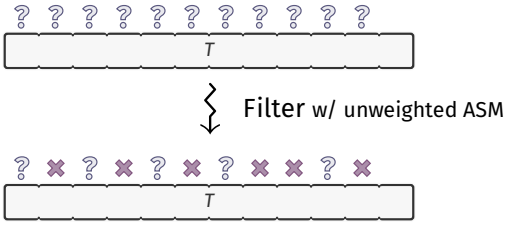
Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)

$\rightsquigarrow$ Use unweighted $O(kn)$-time ASM as Filter



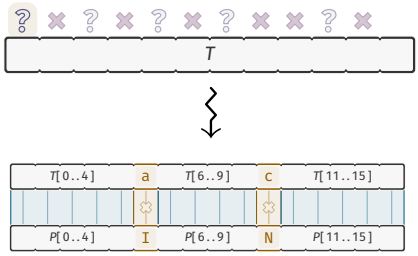
Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)
$\rightsquigarrow$ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[\,a\,..\,b\,]) \leq k$, also get cheap alignment
$P \rightsquigarrow\!\!\!\rightarrow T[\,a\,..\,b\,]$ that matches exactly most of P



Weighted Approximate String Matching

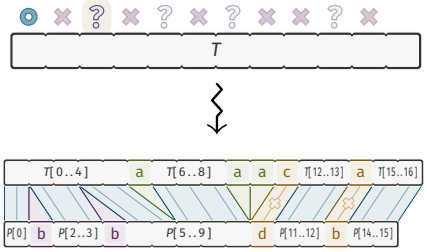
Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)
$\rightsquigarrow$ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[\,a\,..\,b\,]) \leq k$, also get cheap alignment
$P \rightsquigarrow\!\!\!\gg T[\,a\,..\,b\,]$ that matches exactly most of P

All such alignments very similar to each other
But how to exploit this fact?



Weighted Approximate String Matching

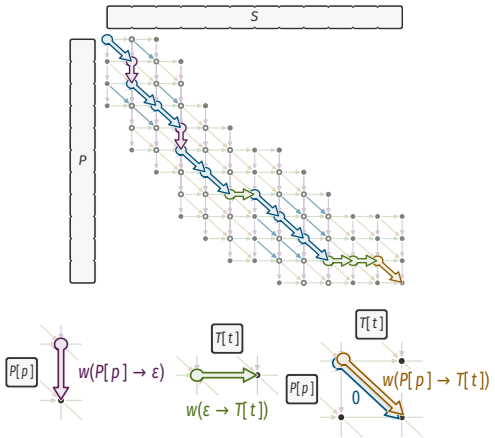
Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)
$\rightsquigarrow$ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[\,a\,..\,b\,]) \leq k$, also get cheap alignment
$P \rightsquigarrow\!\!\!\gg T[\,a\,..\,b\,]$ that matches exactly most of P

All such alignments very similar to each other
But how to exploit this fact?
$\rightsquigarrow$ Cast as shortest path prob in alignment graphs



Weighted Approximate String Matching

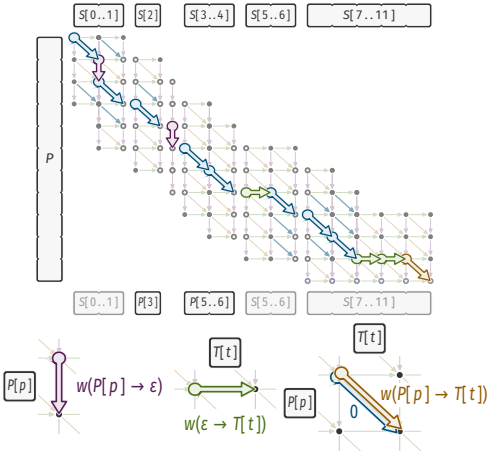
Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)
$\rightsquigarrow$ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[\,a\,..\,b\,]) \leq k$, also get cheap alignment
$P \rightsquigarrow\!\!\!\rightarrow T[\,a\,..\,b\,]$ that matches exactly most of P

All such alignments very similar to each other
But how to exploit this fact?
$\rightsquigarrow$ Cast as shortest path prob in alignment graphs
$\rightsquigarrow$ Copy matched parts from (trivial) align. $P \rightsquigarrow\!\!\!\rightarrow P$



Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

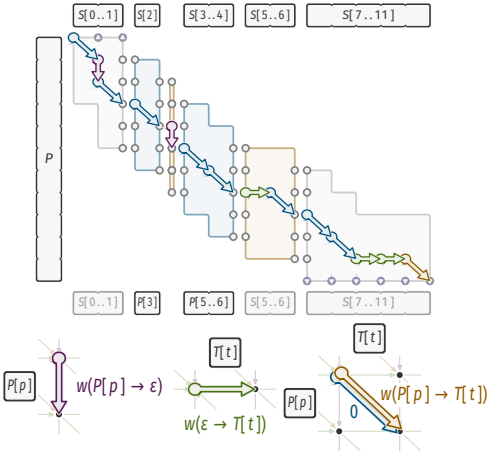
Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)
$\rightsquigarrow$ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[\,a\,..\,b\,]) \leq k$, also get cheap alignment
$P \rightsquigarrow\!\!\!\rightarrow T[\,a\,..\,b\,]$ that matches exactly most of P

All such alignments very similar to each other
But how to exploit this fact?
$\rightsquigarrow$ Cast as shortest path prob in alignment graphs
$\rightsquigarrow$ Copy matched parts from (trivial) align. $P \rightsquigarrow\!\!\!\rightarrow P$

Comp. bound-to-bound dist for each block [Klein'05];
then combine using fast (min, +)-product [SMAWK]



Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

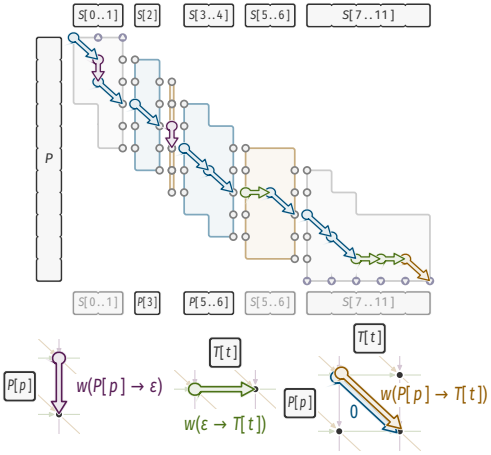
Focus: Obtain starting positions of occurrences

Observation: $\mathrm{ED}^w(X, Y) \geq \mathrm{ED}(X, Y)$ (by norm.)
$\rightsquigarrow$ Use unweighted $O(kn)$-time ASM as Filter

If $\mathrm{ED}(P, T[\,a\,..\,b\,]) \leq k$, also get cheap alignment
$P \rightsquigarrow\!\!\!\rightarrow T[\,a\,..\,b\,]$ that matches exactly most of P

All such alignments very similar to each other
But how to exploit this fact?
$\rightsquigarrow$ Cast as shortest path prob in alignment graphs
$\rightsquigarrow$ Copy matched parts from (trivial) align. $P \rightsquigarrow\!\!\!\rightarrow P$

Comp. bound-to-bound dist for each block [Klein'05];
then combine using fast (min, +)-product [SMAWK]
Precompute all P vs P blocks [Klein'05];
$\rightsquigarrow$ Verify $\leq k$ pos in $\tilde{O}(k^2)$ after $\tilde{O}(kp)$ prep. $\rightsquigarrow$ $\tilde{O}(kt)$ total



Weighted ASM: Bottom Lines

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Main Theorem [ChKoW'25]

Weighted ASM is in time $\tilde{O}(t + t/p \cdot k^4)$ and in $O(k^4)$ PILLAR time for $t < {}^3/_2\, p + k$.

Main Theorem [ChKoW'25]

Weighted ASM is in time $\tilde{O}(kt)$.

Weighted ASM: Bottom Lines

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Main Theorem [ChKoW'25]

Weighted ASM is in time $\tilde{O}(t + t/p \cdot k^4)$ and in $O(k^4)$ PILLAR time for $t < {}^3/_2\, p + k$.

Main Theorem [ChKoW'25]

Weighted ASM is in time $\tilde{O}(kt)$.

Generalization of PILLAR ASM algorithm to weighted setting "easy".

Weighted Approximate String Matching

Given: T, P, k, oracle to w; Find: (starting pos. of) substr. of T at weighted ED $\leq k$ to P.

Main Theorem [ChKoW'25]

Weighted ASM is in time $\tilde{O}(t + t/p \cdot k^4)$ and in $O(k^4)$ PILLAR time for $t < {}^3/_2\, p + k$.

Main Theorem [ChKoW'25]

Weighted ASM is in time $\tilde{O}(kt)$.

Generalization of PILLAR ASM algorithm to weighted setting "easy".

But, also interesting non-PILLAR algorithms out there.

Open Problems and Future Directions

- Big Question: Close gap between UB $O(n + k^{3.5})$ / LB $\Omega(n + k^2)$ for ASM.
  - An $\tilde{O}(k^2)$-time PILLAR algo would imply optimal algorithms in many other settings (e.g. quantum, compressed, etc.)
  - First step: $\tilde{O}(k^3)$-time PILLAR algo
  - Stronger (conditional) LB would require new techniques
  - Similar gap in weighted setting

- Big Question: Close gap between UB $O(n + k^{3.5})$ / LB $\Omega(n + k^2)$ for ASM.
  - An $\tilde{O}(k^2)$-time PILLAR algo would imply optimal algorithms in many other settings (e.g. quantum, compressed, etc.)
  - First step: $\tilde{O}(k^3)$-time PILLAR algo
  - Stronger (conditional) LB would require new techniques
  - Similar gap in weighted setting
- Big Question: Reporting substrings instead of starting positions
  - Alignment graph-based methods seem to fare better
  - First step: $k^{4-\varepsilon}$-time PILLAR algo for ASM
  - First step: $\tilde{O}(kp)$-time for WASM

- Big Question: Close gap between UB $O(n + k^{3.5})$ / LB $\Omega(n + k^2)$ for ASM.
  - An $\tilde{O}(k^2)$-time PILLAR algo would imply optimal algorithms in many other settings (e.g. quantum, compressed, etc.)
  - First step: $\tilde{O}(k^3)$-time PILLAR algo
  - Stronger (conditional) LB would require new techniques
  - Similar gap in weighted setting
- Big Question: Reporting substrings instead of starting positions
  - Alignment graph-based methods seem to fare better
  - First step: $k^{4-\varepsilon}$-time PILLAR algo for ASM
  - First step: $\tilde{O}(kp)$-time for WASM
- Big Question: Simpler algorithms with similar guarantees
  - Especially almost periodic case highly involved
  - Structural insights currently come with impractical constants

Open Problems and Future Directions

- Big Question: Close gap between UB $O(n + k^{3.5})$ / LB $\Omega(n + k^2)$ for ASM.
  - An $\tilde{O}(k^2)$-time PILLAR algo would imply optimal algorithms in many other settings (e.g. quantum, compressed, etc.)
  - First step: $\tilde{O}(k^3)$-time PILLAR algo
  - Stronger (conditional) LB would require new techniques
  - Similar gap in weighted setting
- Big Question: Reporting substrings instead of starting positions
  - Alignment graph-based methods seem to fare better
  - First step: $k^{4-\varepsilon}$-time PILLAR algo for ASM
  - First step: $\tilde{O}(kp)$-time for WASM
- Big Question: Simpler algorithms with similar guarantees
  - Especially almost periodic case highly involved
  - Structural insights currently come with impractical constants
- Big Question: Generalizing idea behind PILLAR to other problem families, settings

Start
Contents
End