From Strings to Seaweeds
Modern Tools for Classical Problems

Philip Wellnitz
National Institute of Informatics
Tokyo

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)

...

...

(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)

...

…

(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, ...

Approximate String Matching

early 1980's

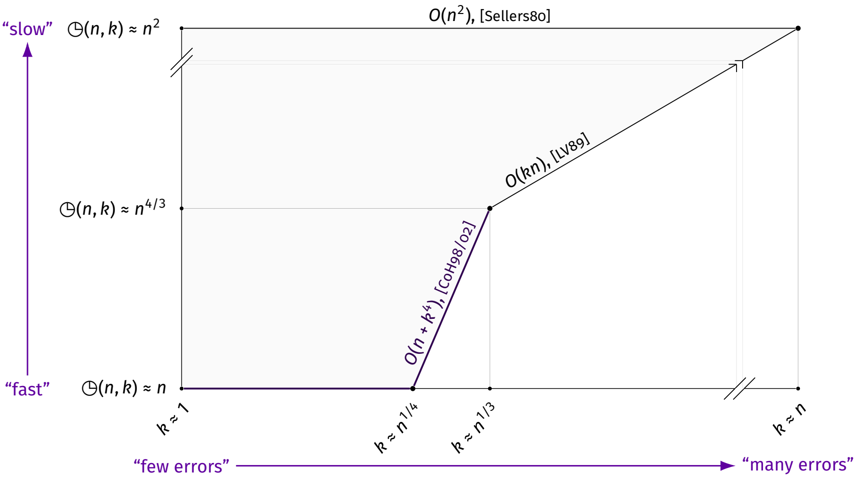
Given a text T, a pattern string P, and an integer k, identify the (starting positions of) substrings of T that are at **edit distance** of at most k to P.

early 1980's

Approximate String Matching

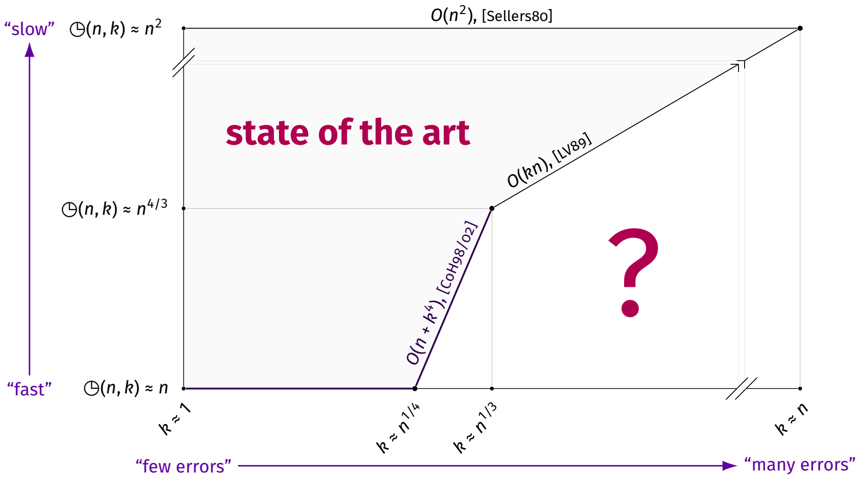
Given a text T, a pattern string P, and an integer k, identify the (starting positions of) substrings of T that are at **edit distance** of at most k to P.

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



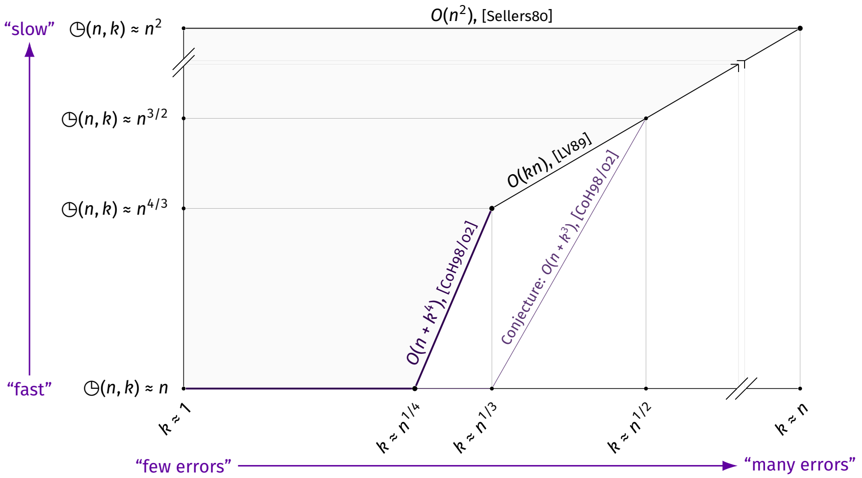
Existing algorithms for the case $n \approx t \approx p$

State-of-the-Art Algorithms



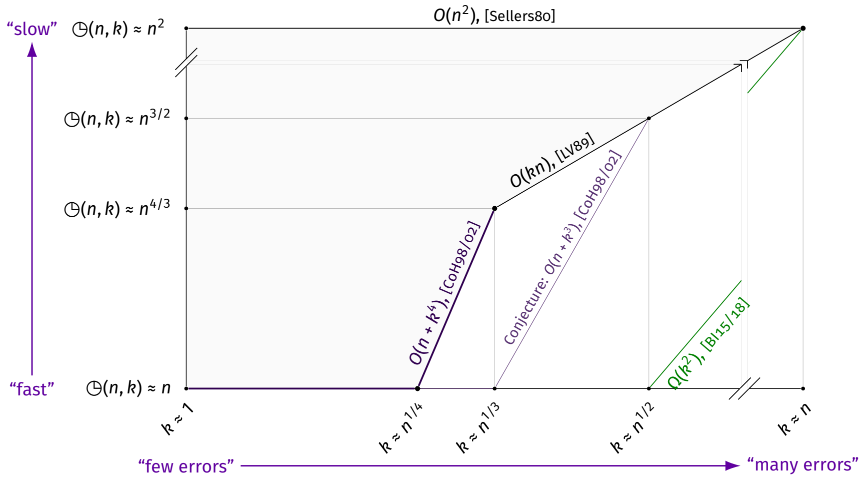
Existing algorithms for the case $n \approx t \approx p$

State-of-the-Art Algorithms

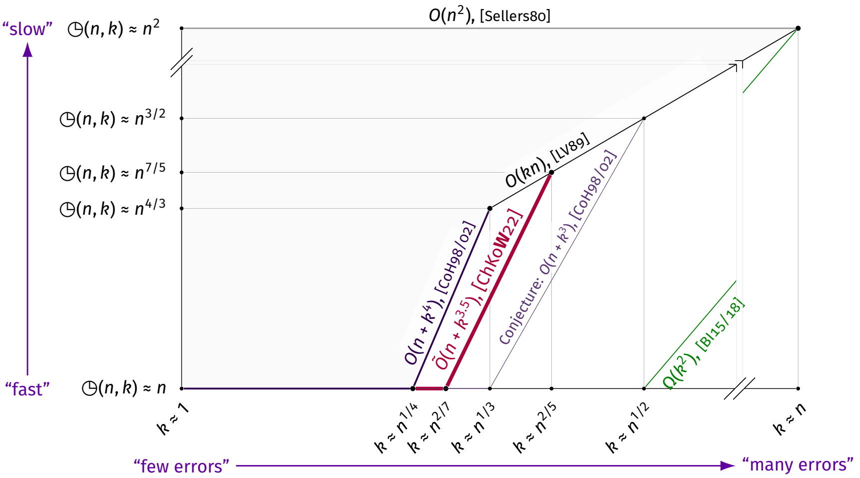


Existing algorithms for the case $n \approx t \approx p$

State-of-the-Art Algorithms



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



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

Intro
- Some Background and Basic Definitions
- (Some) Classical Algorithms for Approximate String Matching

Structural Insights into the Solution Structure
- The Marking Trick and How to Handle Easy Patterns
- Dynamic Puzzle Matching and How to Handle Not So Easy Patterns

On Puzzles and Seaweeds
- Some Technical Definitions
- Detour: On the Name of Seaweeds
- Alignment Graphs and Permutation Matrices of Strings
- Solving Dynamic Puzzle Matching via the Seaweed Method

Outlook: Pattern Matching with Weighted Edits and Open Problems

Approximate String Matching

For 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.

- Naive idea: Compute edit distance to pattern starting from any position in the text
  $\rightsquigarrow$ Need to compute edit distance between strings

Approximate String Matching

For 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.

- Naive idea: Compute edit distance to pattern starting from any position in the text
  $\rightsquigarrow$ Need to compute edit distance between strings

Edit Distance (ED)

For two strings T and P, compute their edit distance
(minimum number of insertions, deletions, and substitutions to transform T into P).

- Textbook dynamic programming algorithm from around 1970, runs in time $O(tp)$
  (writing $t = |T|$, $p = |P|$)

Edit Distance (ED)

For two strings T and P, compute their edit distance
(minimum number of insertions, deletions, and substitutions to transform T into P).

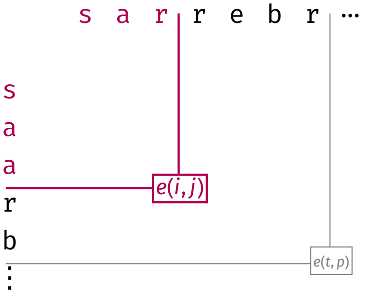
- Textbook dynamic programming algorithm from around 1970, runs in time $O(tp)$
  (writing $t = |T|$, $p = |P|$)

Edit Distance (ED)

For two strings T and P, compute their edit distance
(minimum number of insertions, deletions, and substitutions to transform T into P).

- Textbook dynamic programming algorithm from around 1970, runs in time $O(tp)$
  (writing $t = |T|$, $p = |P|$)
- Write $e(i,j)$ for the ED of $T[0..i)$ and $P[0..j)$
  $\rightsquigarrow$ $e(t,p)$ is the ED of T and P.
- Clearly, $e(i,0) = i$ (delete $T[0..i)$) and $e(0,j) = j$ (insert $P[0..j)$)
- Observe

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



Edit Distance (ED)

For two strings T and P, compute their edit distance
(minimum number of insertions, deletions, and substitutions to transform T into P).

- Textbook dynamic programming algorithm from around 1970, runs in time $O(tp)$ (writing $t = |T|$, $p = |P|$)
- Write $e(i,j)$ for the ED of $T[\,0\,..\,i\,)$ and $P[\,0\,..\,j\,)$
  $\rightsquigarrow$ $e(t,p)$ is the ED of T and P.
- Clearly, $e(i,0) = i$ (delete $T[\,0\,..\,i\,)$) and $e(0,j) = j$ (insert $P[\,0\,..\,j\,)$)
- Observe

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

		s	a	r	r	e	b	r	···
	0	1	2	3	4	5	6	7	
s	1								
a	2								
a	3								
r	4								
b	5								
⋮									

Edit Distance (ED)

For two strings T and P, compute their edit distance
(minimum number of insertions, deletions, and substitutions to transform T into P).

- Textbook dynamic programming algorithm from around 1970, runs in time $O(tp)$
  (writing $t = |T|$, $p = |P|$)
- Write $e(i, j)$ for the ED of $T[\,0\,..\,i\,)$ and $P[\,0\,..\,j\,)$
  $\rightsquigarrow$ $e(t, p)$ is the ED of T and P.
- Clearly, $e(i, 0) = i$ (delete $T[\,0\,..\,i\,)$)
  and $e(0, j) = j$ (insert $P[\,0\,..\,j\,)$)
- Observe

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

		s	a	r	r	e	b	r	…
	0	1	2	3	4	5	6	7	
s	1	0	1	2	3	4	5	6	
a	2	1	0	1	2	3	4	5	
a	3	2	1						
r	4								
b	5								
⋮									

Edit Distance (ED)

For two strings T and P, compute their edit distance
(minimum number of insertions, deletions, and substitutions to transform T into P).

- Textbook dynamic programming algorithm from around 1970, runs in time $O(tp)$ (writing $t = |T|$, $p = |P|$)
- Write $e(i,j)$ for the ED of $T[0..i)$ and $P[0..j)$
  $\rightsquigarrow$ $e(t,p)$ is the ED of T and P.
- Clearly, $e(i,0) = i$ (delete $T[0..i)$) and $e(0,j) = j$ (insert $P[0..j)$)
- Observe

$$e(i,j) = \min \begin{cases} e(i-1,j)+1 & \text{(delete from } T\text{)} \\ e(i,j-1)+1 & \text{(insert in } T\text{)} \\ e(i-1,j-1) & \\ \quad +(T[i] \neq P[j]) & \text{(match/}\textbf{subst}\text{)} \end{cases}$$

		s	a	r	r	e	b	r	⋯
	0	1	2	3	4	5	6	7	
s	1	0	1	2	3	4	5	6	
a	2	1	0	1	2	3	4	5	
a	3	2	1	$\boxed{1}$					
r	4								
b	5								
⋮									

Edit Distance (ED)

For two strings T and P, compute their edit distance
(minimum number of insertions, deletions, and substitutions to transform T into P).

- Textbook dynamic programming algorithm from around 1970, runs in time $O(tp)$
  (writing $t = |T|$, $p = |P|$)
- Write $e(i,j)$ for the ED of $T[\,0\,..\,i\,)$ and $P[\,0\,..\,j\,)$
  $\rightsquigarrow$ $e(t,p)$ is the ED of T and P.
- Clearly, $e(i,0) = i$ (delete $T[\,0\,..\,i\,)$) and $e(0,j) = j$ (insert $P[\,0\,..\,j\,)$)
- Observe

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

		s	a	r	r	e	b	r	…
	0	1	2	3	4	5	6	7	
s	1	0	1	2	3	4	5	6	
a	2	1	0	1	2	3	4	5	
a	3	2	1	1	$\boxed{2}$				
r	4								
b	5								
⋮									

Edit Distance (ED)

For two strings T and P, compute their edit distance
(minimum number of insertions, deletions, and substitutions to transform T into P).

- Textbook dynamic programming algorithm from around 1970, runs in time $O(tp)$
  (writing $t = |T|$, $p = |P|$)
- Write $e(i,j)$ for the ED of $T[0..i)$ and $P[0..j)$
  $\rightsquigarrow e(t,p)$ is the ED of T and P.
- Clearly, $e(i,0) = i$ (delete $T[0..i)$) and $e(0,j) = j$ (insert $P[0..j)$)
- Observe

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

		s	a	r	r	e	b	r	…
	0	1	2	3	4	5	6	7	
s	1	0	1	2	3	4	5	6	
a	2	1	0	1	2	3	4	5	
a	3	2	1	1	2	3	4	5	
r	4	3	2	1	1	2	3	4	
b	5	4	3	2	2	2	2	3	
⋮									

Approximate String Matching

For 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.

- Textbook dynamic programming algorithm from ~~~1970~~ 1980, runs in time $O(tp)$
  (writing $t = |T|$, $p = |P|$), [Sellers 1980, rediscovered several times]
- Write $e(i, j)$ for the ED of $T[\,0\,..\,i\,)$ and $P[\,0\,..\,j\,)$
  after deleting an arbitrary prefix of T
- Clearly, $e(i, 0) = 0$ (delete $T[\,0\,..\,i\,)$)
  and $e(0, j) = j$ (insert $P[\,0\,..\,j\,)$)
- Observe

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

$$\begin{array}{c|cccccccc} & \mathtt{s} & \mathtt{a} & \mathtt{r} & \mathtt{r} & \mathtt{e} & \mathtt{b} & \mathtt{r} & \cdots \\ \hline & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathtt{s} & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathtt{a} & 2 & 1 & 0 & 1 & 2 & 2 & 2 & 2 \\ \mathtt{a} & 3 & 2 & 1 & 1 & 2 & 3 & 3 & 3 \\ \mathtt{r} & 4 & 3 & 2 & 1 & 1 & 2 & 3 & 3 \\ \mathtt{b} & 5 & 4 & 3 & 2 & 2 & 2 & 2 & 3 \\ \vdots & & & & & & & & \end{array}$$

Approximate String Matching

For 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.

- Textbook dynamic programming algorithm from ~~~1970~~ 1980, runs in time $O(tp)$ (writing $t = |T|$, $p = |P|$), [Sellers 1980, rediscovered several times]
- Write $e(i,j)$ for the ED of $T[0..i)$ and $P[0..j)$ after deleting an arbitrary prefix of T
- Clearly, $e(i,0) = 0$ (delete $T[0..i)$) and $e(0,j) = j$ (insert $P[0..j)$)
- Observe

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

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

Ending positions of saarb with $k = 2$.

Approximate String Matching

For 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.

- Textbook dynamic programming algorithm from ~~~1970~~ 1980, runs in time $O(tp)$ (writing $t = |T|$, $p = |P|$), [Sellers 1980, rediscovered several times]
- Write $e(i,j)$ for the ED of $T^R[\,0\,..\,i\,)$ and $P^R[\,0\,..\,j\,)$ after deleting an arbitrary prefix of T^R
- Clearly, $e(i,0) = 0$ (delete $T[\,0\,..\,i\,)$) and $e(0,j) = j$ (insert $P[\,0\,..\,j\,)$)
- Observe

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

	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

Starting positions of `saarb` with $k = 2$.

Approximate String Matching

For 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.

(Variations of) Sellers' $O(tp)$ Algorithm were (are?) popular for

- (Have seen) **Text Retrieval**: Find occurrences of patterns or phrases in a text, accounting for misspellings or conversion errors
- **Signal Processing**: Find patterns in signals, accounting for transmission errors
- **Computational Biology**: Find specific patterns in DNA sequences, accounting for mutations or evolutionary alterations

Approximate String Matching

For 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.

(Variations of) Sellers' $O(tp)$ Algorithm were (are?) popular for

- (Have seen) **Text Retrieval**: Find occurrences of patterns or phrases in a text, accounting for misspellings or conversion errors
- **Signal Processing**: Find patterns in signals, accounting for transmission errors
- **Computational Biology**: Find specific patterns in DNA sequences, accounting for mutations or evolutionary alterations

There must be faster algorithms out there!

There must be faster algorithms out there!

There must be faster algorithms out there!

Approximate String Matching

For 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.

- So far, did not use k
  - $\leadsto$ Do not need values $e(i,j) > k$
- How do we compute just values $\le k$?
  - $\leadsto$ Compute values diagonally.

		r	r	a	s	r	a	a	s	r	h	a	s	r	r	a	s	…
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
r	1	0	0	1	1	0	1	1	1	0	1	1	1	0	0	1	1	
a	2	1	1	0	1	1	0	1	2	1	1	1	2	1	1	0	1	
a	3	2	2	1	1	2	1	0	1	2	2	1	2	2	2	1	1	
s	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2	1	
r	5	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2	
a	6	5	4	3	3	2	1	2	2	1	1	1	2	2	2	2	3	
a	7	6	5	4	4	3	2	1	2	2	2	1	2	3	3	2	3	
s	8	7	6	5	4	4	3	2	1	2	3	2	1	2	3	3	2	

Finding `saarsaar` with $k \le 2$ edits.

Approximate String Matching

For 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.

- So far, did not use k
  $\rightsquigarrow$ Do not need values $e(i,j) > k$
- How do we compute just values $\leq k$?
  $\rightsquigarrow$ Compute values diagonally.

		r	r	a	s	r	a	a	s	r	h	a	s	r	r	a	s …
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	1	0	0	1	1	0	1	1	1	0	1	1	1	0	0	1	1
a	2	1	1	0	1	1	0	1	2	1	1	1	2	1	1	0	1
a	3	2	2	1	1	2	1	0	1	2	2	1	2	2	2	1	1
s	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2	1
r	5	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2
a	6	5	4	3	3	2	1	2	2	1	1	1	2	2	2	2	3
a	7	6	5	4	4	3	2	1	2	2	2	1	2	3	3	2	3
s	8	7	6	5	4	4	3	2	1	2	3	2	1	2	3	3	2

Finding `saarsaar` with $k \leq 2$ edits.

Using Kangaroo Jumps for Speed-Ups

Approximate String Matching

For 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.

- So far, did not use k
  $\rightsquigarrow$ Do not need values $e(i,j) > k$
- How do we compute just values $\leq k$?
  $\rightsquigarrow$ Compute values diagonally.

		r	r	a	s	r	a	a	s	r	h	a	s	r	r	a	s	…
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
r	1	0	0	1	1	0	1	1	1	0	1	1	1	0	0	1	1	
a	2	1	1	0	1	1	0	1	2	1	1	1	2	1	1	0	1	
a	3	2	2	1	1	2	1	0	1	2	2	1	2	2	2	1	1	
s	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2	1	
r	5	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2	
a	6	5	4	3	3	2	1	2	2	1	1	1	2	2	2	2	3	
a	7	6	5	4	4	3	2	1	2	2	2	1	2	3	3	2	3	
s	8	7	6	5	4	4	3	2	1	2	3	2	1	2	3	3	2	

Finding saarsaar with $k \leq 2$ edits.

Approximate String Matching

For 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.

- So far, did not use k
  ⇝ Do not need values $e(i,j) > k$
- How do we compute just values $\leq k$?
  ⇝ Compute values diagonally.

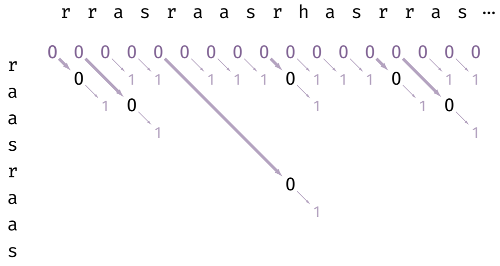
		r	r	a	s	r	a	a	s	r	h	a	s	r	r	a	s	…
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
r	1	0	0	1	1	0	1	1	1	0	1	1	1	0	0	1	1	
a	2	1	1	0	1	1	0	1	2	1	1	1	2	1	1	0	1	
a	3	2	2	1	1	2	1	0	1	2	2	1	2	2	2	1	1	
s	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2	1	
r	5	4	3	3	2	1	2	2	1	0	1	2	2	1	2	3	2	
a	6	5	4	3	3	2	1	2	2	1	1	1	2	2	2	2	3	
a	7	6	5	4	4	3	2	1	2	2	2	1	2	3	3	2	3	
s	8	7	6	5	4	4	3	2	1	2	3	2	1	2	3	3	2	

Finding `saarsaar` with $k \leq 2$ edits.

Approximate String Matching

For 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.

- So far, did not use k
  $\rightsquigarrow$ Do not need values $e(i,j) > k$
- How do we compute just values $\le k$?
  $\rightsquigarrow$ Compute furthest position on diag d reachable with $\ell = 0, \ldots, k$ edits.
  $\rightsquigarrow$ Jump over equal substrings in $O(1)$ time (Kangaroo Jumps).
  $\rightsquigarrow$ In total: $O((t+p)k)$ time time [Landau, Vishkin'89].

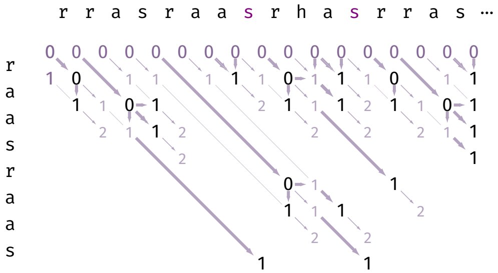


Finding `saarsaar` with $k \le 0$ edits.

Approximate String Matching

For 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.

- So far, did not use k
  $\rightsquigarrow$ Do not need values $e(i, j) > k$
- How do we compute just values $\leq k$?
  $\rightsquigarrow$ Compute furthest position on diag d reachable with $\ell = 0, \dots, k$ edits.
  $\rightsquigarrow$ Jump over equal substrings in $O(1)$ time (Kangaroo Jumps).
  $\rightsquigarrow$ In total: $O((t + p)k)$ time time [Landau, Vishkin'89].

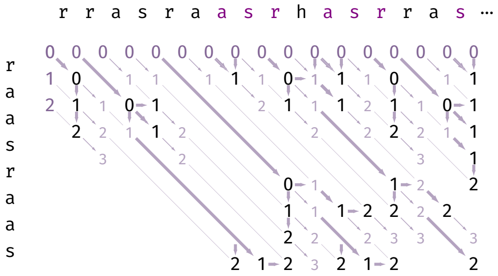


Finding `saarsaar` with $k \leq 1$ edits.

Approximate String Matching

For 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.

- So far, did not use k
  $\rightsquigarrow$ Do not need values $e(i, j) > k$
- How do we compute just values $\leq k$?
  $\rightsquigarrow$ Compute furthest position on diag d reachable with $\ell = 0, \ldots, k$ edits.
  $\rightsquigarrow$ Jump over equal substrings in $O(1)$ time (Kangaroo Jumps).
  $\rightsquigarrow$ In total: $O((t + p)k)$ time time [Landau, Vishkin'89].

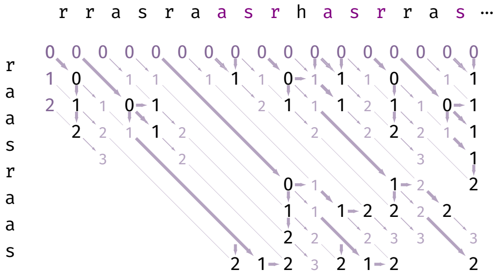


Finding saarsaar with $k \leq 2$ edits.

Approximate String Matching

For 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.

- So far, did not use k
  $\rightsquigarrow$ Do not need values $e(i, j) > k$
- How do we compute just values $\leq k$?
  $\rightsquigarrow$ Compute furthest position on diag d reachable with $\ell = 0, \ldots, k$ edits.
  $\rightsquigarrow$ Jump over equal substrings in $O(1)$ time (Kangaroo Jumps).
  $\rightsquigarrow$ In total: $O((t + p)k)$ time time [Landau, Vishkin'89].

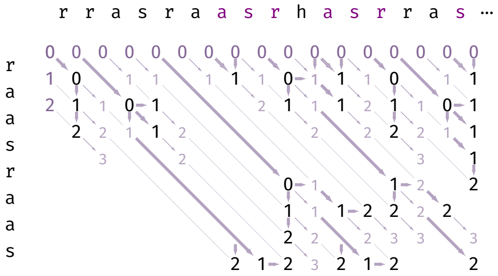


Finding `saarsaar` with $k \leq 2$ edits.

Approximate String Matching

For 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.

- So far, did not use k
  $\rightsquigarrow$ Do not need values $e(i,j) > k$
- How do we compute just values $\leq k$?
  $\rightsquigarrow$ Compute furthest position on diag d reachable with $\ell = 0, \dots, k$ edits.
  $\rightsquigarrow$ Jump over equal substrings in $O(1)$ time (Kangaroo Jumps). ← **complicated**
  $\rightsquigarrow$ In total: $O((t+p)k)$ time time [Landau, Vishkin'89].



Finding `saarsaar` with $k \leq 2$ edits.

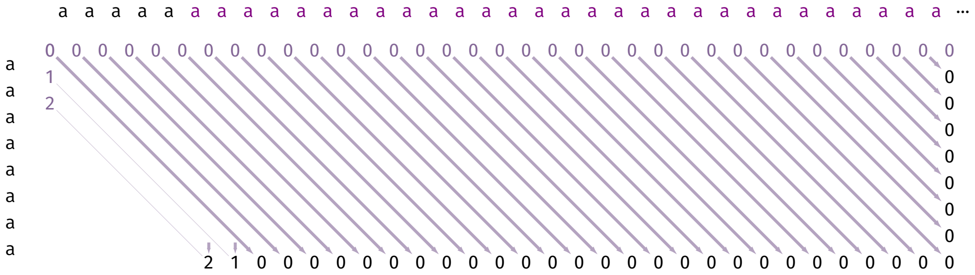
There must be faster algorithms out there!

There must be faster algorithms out there!

...

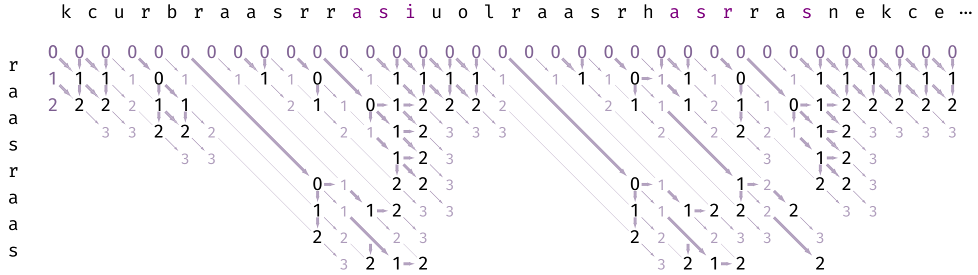
Idea: Can we filter out diagonals that will never lead to an occurrence?

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



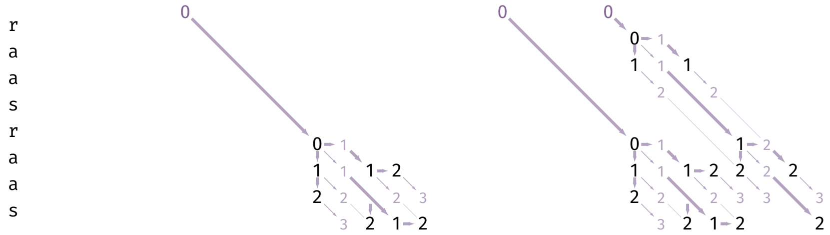
Beating Landau and Vishkin's Algorithm

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!



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!

k c u r b r a a s r r a s i u o l r a a s r h a s r r a s n e k c e …



Approximate String Matching

For 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.

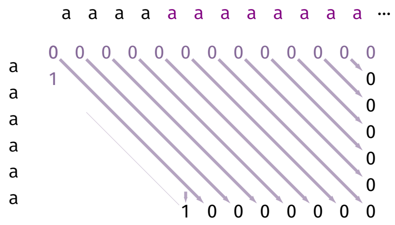
- Idea: Find parts of P that are rare in T
  $\rightsquigarrow$ Can filter out candidates for occurrences
- Seen: Highly repetitive parts R are bad...
  $\rightsquigarrow$ R may appear $\approx t$ times in T
- Non-repetitive parts B are good!
  $\rightsquigarrow$ B may appear only $\approx t/b$ times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

Unstructured Parts of the Pattern Help!

Approximate String Matching

For 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.

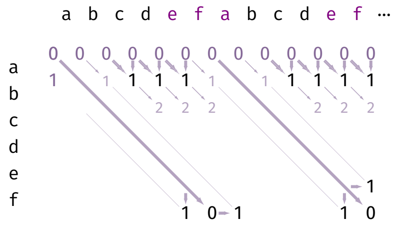
- Idea: Find parts of P that are rare in T
  - $\rightsquigarrow$ Can filter out candidates for occurrences
- Seen: Highly repetitive parts R are bad...
  - $\rightsquigarrow$ R may appear $\approx t$ times in T
- Non-repetitive parts B are good!
  - $\rightsquigarrow$ B may appear only $\approx t/b$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



Approximate String Matching

For 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.

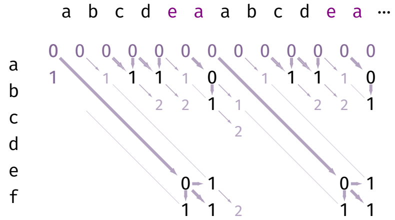
- Idea: Find parts of P that are rare in T
  - ⇝ Can filter out candidates for occurrences
- Seen: Highly repetitive parts R are bad...
  - ⇝ R may appear $\approx t$ times in T
- Non-repetitive parts B are good!
  - ⇝ B may appear only $\approx t/b$ times in T
  - ⇝ Problem: B may appear approximately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



Approximate String Matching

For 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.

- 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

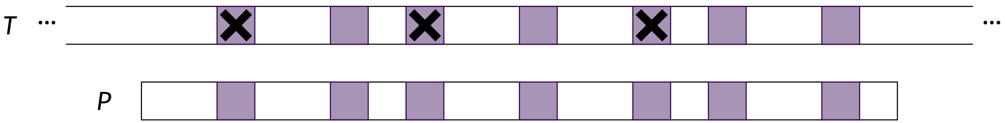


Unstructured Parts of the Pattern Help!

Approximate String Matching

For 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.

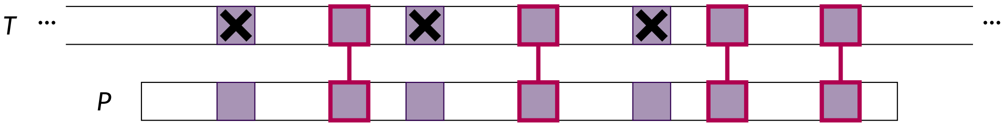
- Idea: Find non-repetitive parts (breaks) B in the pattern $\rightsquigarrow$ B appears rarely in text
  $\rightsquigarrow$ Problem: In an occurrence, B might have an error
- Observation: Out of $2k$ disjoint breaks in P, any occ matches k breaks exactly
  (Have a budget of only k edits in total)
- Leads to $O(t + k^4 \cdot t/p)$ algorithm (more details soon) [Cole,Hariharan'02] (announced '98)



Approximate String Matching

For 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.

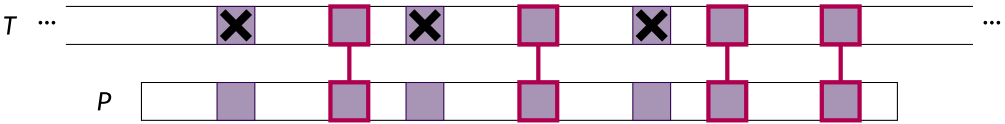
- Idea: Find non-repetitive parts (breaks) B in the pattern $\rightsquigarrow$ B appears rarely in text
  $\rightsquigarrow$ Problem: In an occurrence, B might have an error
- Observation: Out of $2k$ disjoint breaks in P, any occ matches k breaks exactly
  (Have a budget of only k edits in total)
- Leads to $O(t + k^4 \cdot t/p)$ algorithm (more details soon) [Cole,Hariharan'02] (announced '98)

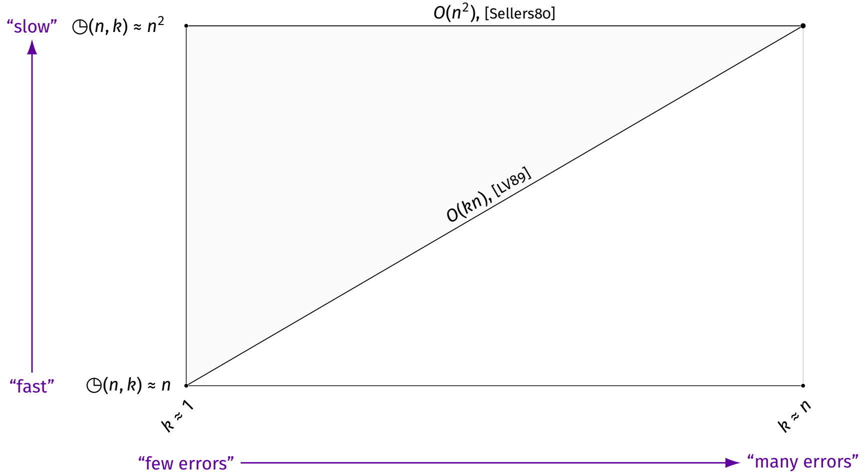


Approximate String Matching

For 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.

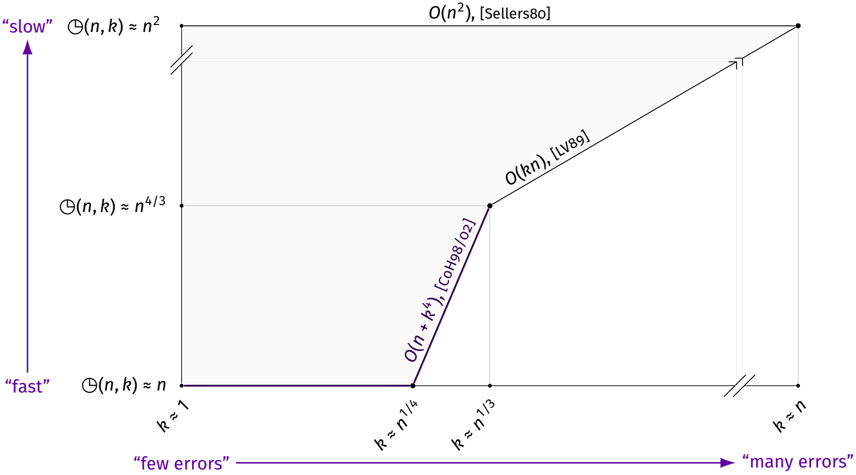
- Idea: Find non-repetitive parts (breaks) B in the pattern $\rightsquigarrow$ B appears rarely in text
  $\rightsquigarrow$ Problem: In an occurrence, B might have an error
- Observation: Out of $2k$ disjoint breaks in P, any occ matches k breaks exactly
  (Have a budget of only k edits in total)
- Leads to $O(t + k^4 \cdot t/p)$ algorithm (more details soon) [Cole,Hariharan'02] (announced '98)





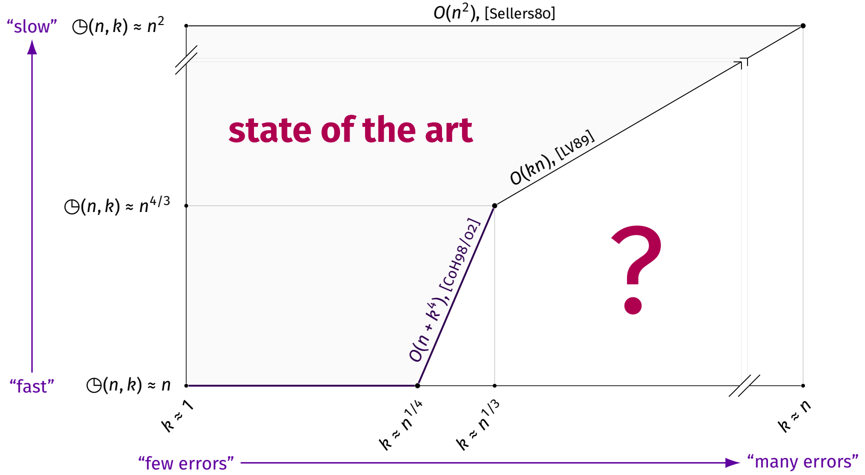
Existing algorithms for the case $n \approx t \approx p$

Recap: State-of-the-Art Algorithms



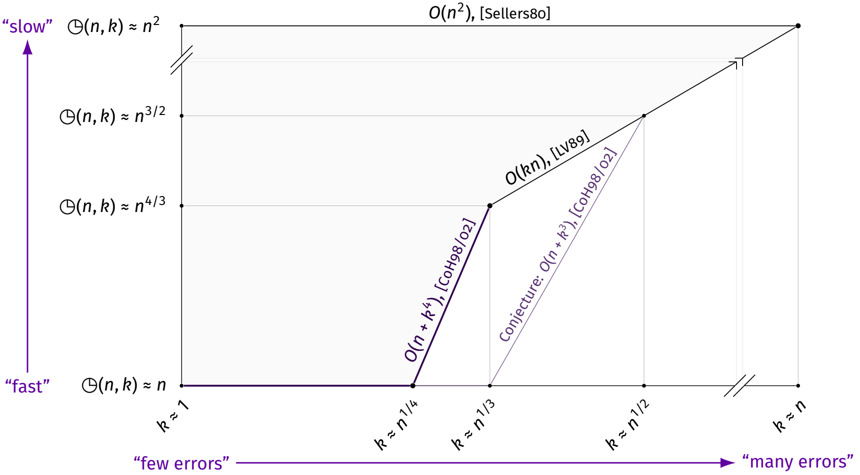
Existing algorithms for the case $n \approx t \approx p$

Recap: State-of-the-Art Algorithms

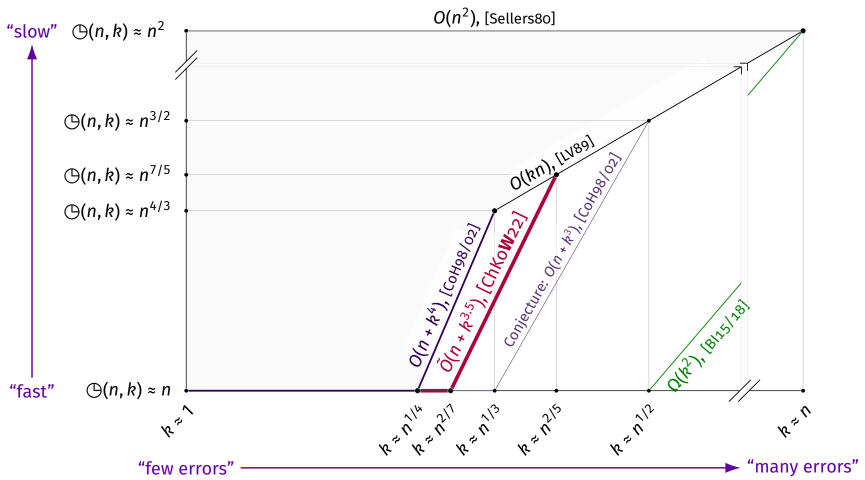


Existing algorithms for the case $n \approx t \approx p$

Recap: State-of-the-Art Algorithms



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



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

Why did a new improvement take 24 years?

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

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

[BI18] (ann. 2014)

Hardness of Edit Distance

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

[BI18] (ann. 2014)

Hardness of Edit Distance

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.

- 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**]

- 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**]

Optimal String Matching with Mismatches [GU18] (ann. 2017)

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)
  - $\leadsto$ $O(tk)$ algorithm [Landau, Vishkin'**86**]
  - $\leadsto$ $\tilde{O}(t\sqrt{k})$ algorithm [Amir, Lewenstein, Porat'**04**]
  - $\leadsto$ $\tilde{O}(t + t/p \cdot k^2)$ algorithm [CFPSS'**16**]

Optimal String Matching with Mismatches [GU18] (ann. 2017)

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.)

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

How do we obtain faster algorithms?

New insights into the solution structure of
Approximate String Matching

How do we obtain faster algorithms?

New insights into the solution structure of Approximate String Matching

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: $\mathrm{per}(S) \le |S|/2$

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

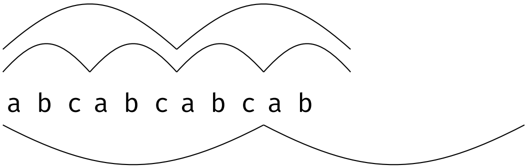
$\mathrm{per}(S) = 3$

(The) Period of a String

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

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

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



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

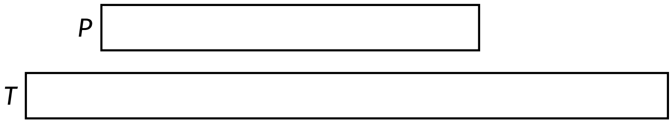
$\mathrm{per}(S) = 3$

6 and 9 also periods of S

"Periodicity Lemma" (Folklore)

Text T, pattern P with $|T| \leq {}^{3}/_{2}\,|P|$; one of the following holds

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

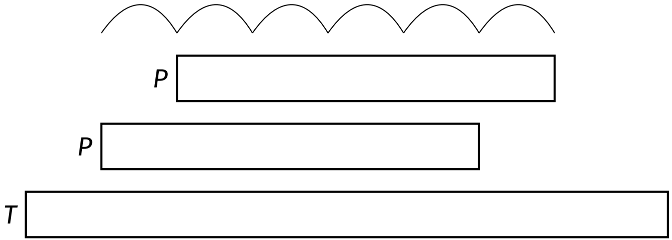


"Periodicity Lemma"

(Folklore)

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

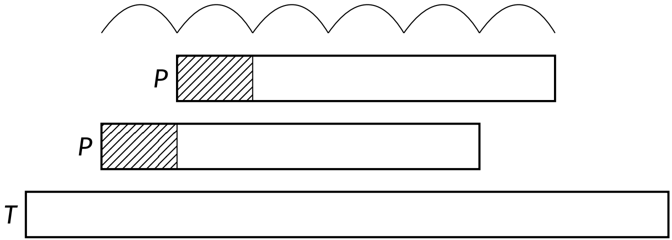
- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



"Periodicity Lemma" (Folklore)

Text T, pattern P with $|T| \le {}^3\!/_2\,|P|$; one of the following holds

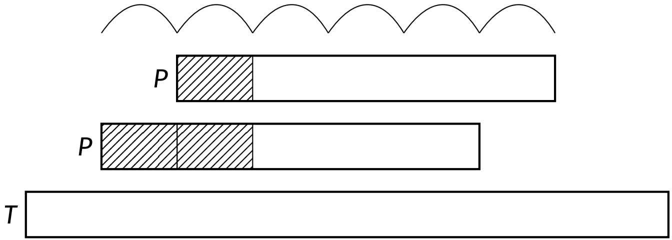
- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



"Periodicity Lemma" (Folklore)

Text T, pattern P with $|T| \leq {}^3\!/_2\,|P|$; one of the following holds

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

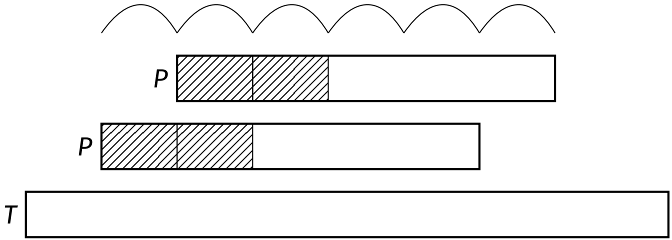


(Folklore)

"Periodicity Lemma"

Text T, pattern P with $|T| \le {}^3/_2\,|P|$; one of the following holds

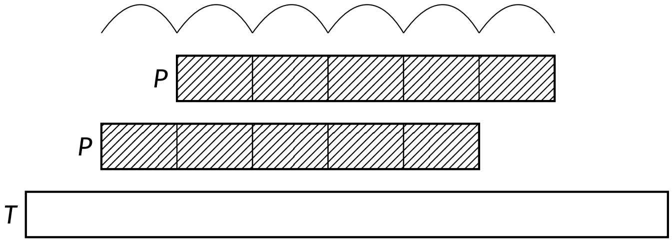
- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



“Periodicity Lemma” (Folklore)

Text T, pattern P with $|T| \le {}^3\!/_2\, |P|$; one of the following holds

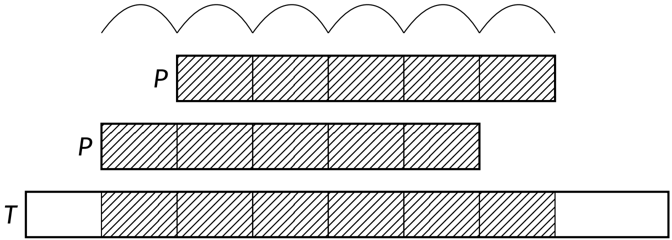
- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



"Periodicity Lemma" (Folklore)

Text T, pattern P with $|T| \leq {}^{3}/_{2}\,|P|$; one of the following holds

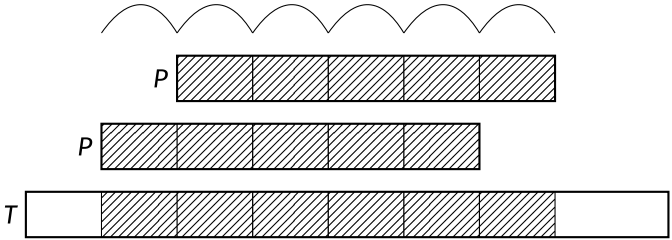
- P appears ≤ 1 times in T.
- P (and the relevant part of T) are periodic with some period Q.



“Periodicity Lemma” (Folklore)

Text T, pattern P with $|T| \leq {}^3\!/_2\,|P|$; one of the following holds

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



The Standard Trick: For $|T| \gg |P|$ consider separately $O(n/m)$ fragments of T of length $\leq {}^3\!/_2\, m$ that overlap by $m - 1$ positions.

Step 1:

What is the solution structure of
String Matching with Mismatches?

Periodicity Lemma

(Folklore)

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

Main Result (Mismatches)

[BKüW'19; CKoW'20]

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

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

T a ⋯ a ⋯ a | c ⋯ c ⋯ c

P a ⋯ a | c ⋯ c

a⋯ac⋯c appears $2k$ times

T a ⋯ a | c | a a a ⋯ a | c

P a a ⋯ a | c | a

aa⋯aca has approximate period a

(is at HD $\le 2k$ from Q^∞)

Periodicity Lemma

(Folklore)

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

Main Result (Mismatches)

[BKüW'19; CKoW'20]

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

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

T	a ⋯ a ⋯ a | c ⋯ c ⋯ c
P	a ⋯ a | c ⋯ c

a⋯ac⋯c appears $2k$ times

T	a ⋯ a | c | a a a ⋯ a | c
P	a a ⋯ a | c | a

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

Periodicity Lemma

(Folklore)

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

Main Result (Mismatches)

[BKüW'19; CKoW'20]

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

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

T | a ⋯ a ⋯ a | c ⋯ c ⋯ c |

P | a ⋯ a | c ⋯ c |

a⋯ac⋯c appears $2k$ times

T | a ⋯ a | c | a a a ⋯ a | c |

P | a a ⋯ a | c | a |

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

Structural Results for Approximate String Matching

Periodicity Lemma
(Folklore)

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

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

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

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

T a ⋯ a ⋯ a | c ⋯ c ⋯ c

P a ⋯ a | c ⋯ c

a⋯ac⋯c appears $2k$ times

T a ⋯ a | c | a a a ⋯ a | c

P a a ⋯ a | c | a

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

Step 1.5:

The solution structure of
Approximate String Matching.

[BKüW'19; CKoW'20]

Main Result (Mismatches)

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

[CKoW'20]

Main Result (Edits)

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

Main Result (Mismatches)

[BKüW'19; CKoW'20]

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

Main Result (Edits)

[CKoW'20]

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

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

T: a a ⋯ a a a ⋯ a a a ⋯ a c a ⋯ a ⋯ c a ⋯ a c a ⋯ a

P: a a ⋯ a a a ⋯ a c a ⋯ a ⋯ c a ⋯ a

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

Structural Results for Approximate String Matching

Main Result (Mismatches)

[BKüW'19; CKoW'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)

[CKoW'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

T	a	a	⋯	a	a	a	⋯	a	a	a	⋯	a	c	a	⋯	a	⋯	c	a	⋯	a	c	a	⋯	a
P	a	a	⋯	a	a	a	⋯	a	c	a	⋯	a	c	a	⋯	a									

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

Main Result (Mismatches)

[BKüW'19; CKoW'20]

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

Main Result (Edits)

[CKoW'20]

Text T, pattern P with $t \le {}^{3}\!/_{2}\, p$; threshold k, one of the following holds

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

T: a a ⋯ a a a ⋯ a a a ⋯ a c a ⋯ a ⋯ c a ⋯ a c a ⋯ a

P: a ⋯ a a a ⋯ a c a ⋯ a c a ⋯ a

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

[CKoW'20]

Main Result (Edits)

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)

[CKoW'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)

[CKoW'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

Structural Results for Approximate String Matching

[CKoW'20]

Main Result (Edits)

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

[CKoW'20]

Key Intermediate Result (Analyze)

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. aaacaaaaaaaacaaaaaaa

Main Result (Edits)

[CKoW'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)

[CKoW'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)

[CKoW'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)

[CKoW'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

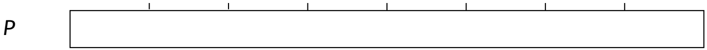
Analyze implies Main Result.

Key Intermediate Result (Analyze)

[CKoW'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



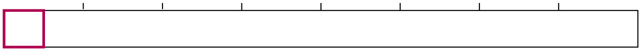
Key Intermediate Result (Analyze)

[CKoW'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

P



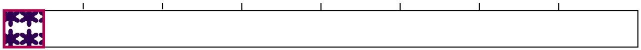
- Process P from left to right, $p/8k$ new characters at a time.

Key Intermediate Result (Analyze)

[CKoW'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

P 

Breaks B = {  }

Repetitive Regions R = { }

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

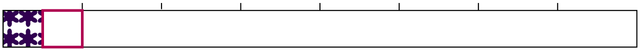
Key Intermediate Result (Analyze)

[CKoW'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

P



Breaks B = {  }

Repetitive Regions R = { }

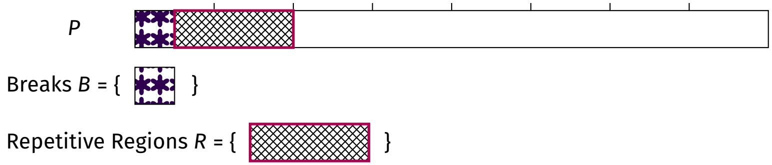
- Otherwise, find the shortest prefix (longer than $p/8k$) that is a repetitive region.

Key Intermediate Result (Analyze)

[CKoW'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. — aaacaaaaaaaacaaaaaaa



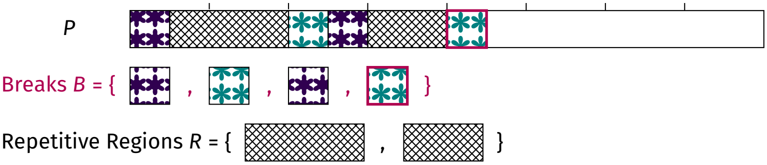
- Otherwise, find the shortest prefix (longer than $p/8k$) that is a repetitive region.

Key Intermediate Result (Analyze)

[CKoW'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



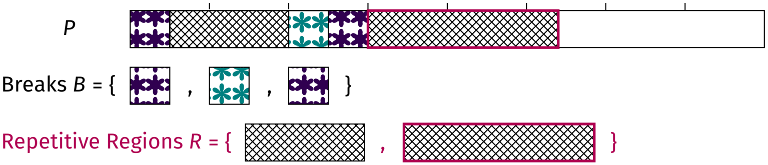
- If we found $2k$ breaks, return the breaks.

[CKoW'20]

Key Intermediate Result (Analyze)

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



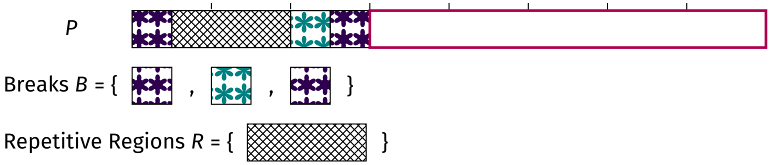
- If the total length of the repetitive regions is $> 3/8 \cdot p$, return the repetitive regions.

Key Intermediate Result (Analyze)

[CKoW'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. — aaacaaaaaaaacaaaaaaa



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

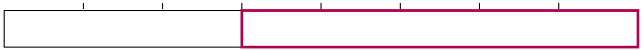
[CKoW'20]

Key Intermediate Result (Analyze)

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. — aaacaaaaaaaacaaaaaaa

P



Breaks $B = \{\ \}$

Repetitive Regions $R = \{\ \}$

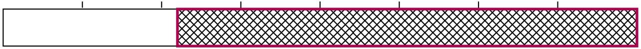
- If we reach the end of P, try to find a single repetitive region starting from the end.

Key Intermediate Result (Analyze)

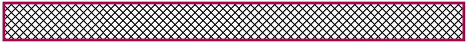
[CKoW'20]

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. — aaacaaaaaaaacaaaaaa



Breaks B = { }

Repetitive Regions R = {  }

- If we found a repetitive region, return it.

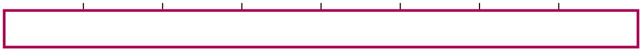
[CKoW'20]

Key Intermediate Result (Analyze)

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

P



Breaks $B = \{\quad\}$

Repetitive Regions $R = \{\quad\}$

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

[CKoW'20]

Key Intermediate Result (Analyze) ✓

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. — aaacaaaaaaaacaaaaaaa

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

Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

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

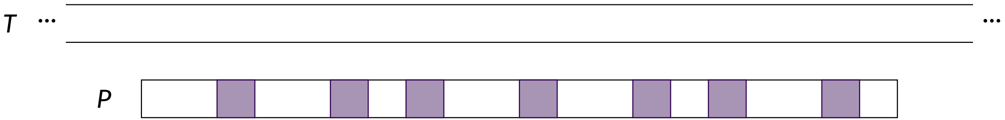
- In any k-edit occ, at least 1 break is matched exactly.

P

Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

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

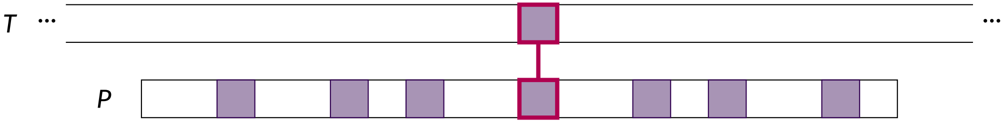
- In any k-edit occ, at least 1 break is matched exactly.



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

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

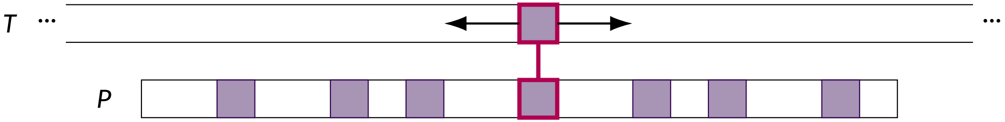
- In any k-edit occ, at least 1 break is matched exactly.
- Algorithm idea: For each B_i, find exact matches of B_i in T



Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

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

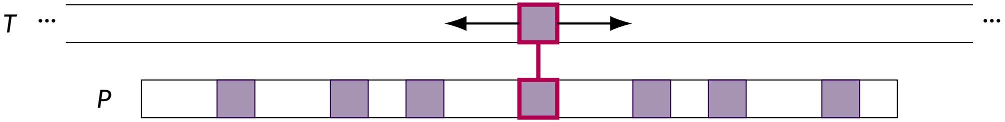
- In any k-edit occ, at least 1 break is matched exactly.
- Algorithm idea: For each B_i, find exact matches of B_i in T,
  Try to extend each exact match into an occ using [LV'89]



Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

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

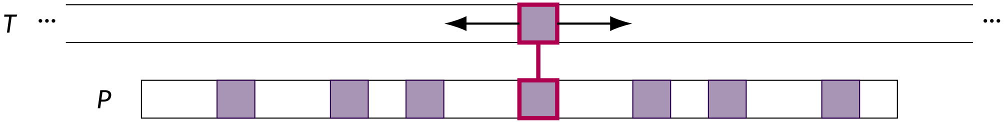
- In any k-edit occ, at least 1 break is matched exactly.
- Algorithm idea: For each B_i, find exact matches of B_i in T,
  Try to extend each exact match into an occ using [LV'89]
- Have at most $|T|/(|B_i|/2) = \Theta(k|T|/|P|)$ exact occ's of B_i
  $\rightsquigarrow O(k^2|T|/|P|)$ calls to [LV'89];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



Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ 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.
- Algorithm idea: For each B_i, find exact matches of B_i in T, Try to extend each exact match into an occ using [LV'89]
- Have at most $|T|/(|B_i|/2) = \Theta(k|T|/|P|)$ exact occ's of B_i $\rightsquigarrow O(k^2|T|/|P|)$ calls to [LV'89]; $O(k^4|T|/|P|)$ time in total



Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

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

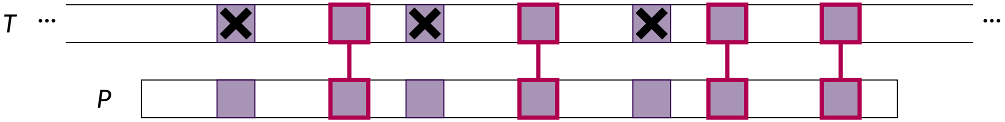
- In any k-edit occ, at least k breaks are matched exactly.
- Algorithm idea: For each B_i, find exact matches of B_i in T

Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ 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.
- Algorithm idea: For each B_i, find exact matches of B_i in T

  Run [LV'89] only for positions in T where at least k breaks match exactly $\rightsquigarrow$ How?

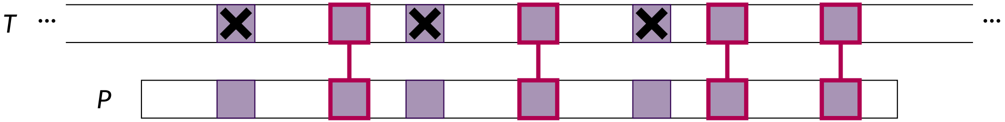


Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

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

- In any k-edit occ, at least k breaks are matched exactly.
- Algorithm idea: For each B_i, find exact matches of B_i in T

  Run [LV'89] only for positions in T where at least k breaks match exactly $\rightsquigarrow$ How?

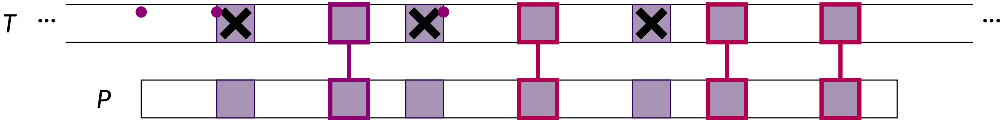


Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

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

- In any k-edit occ, at least k breaks are matched exactly.
- Algorithm idea: For each B_i, find exact matches of B_i in T

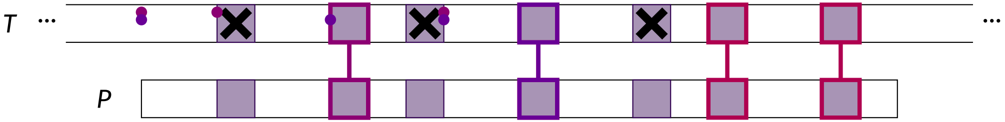
  Run [LV'89] only for positions in T where at least k breaks match exactly $\rightsquigarrow$ How?
- If $B_i = P[\ell_i \,..\,) = T[a \,..\,)$, add a mark to $T[\lfloor (a - \ell_i)/k \rfloor]$; run [LV'89] for pos w/ $\geq k$ marks



Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

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

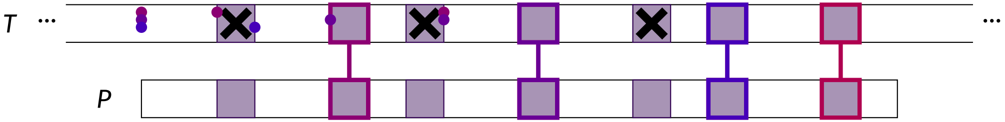
- In any k-edit occ, at least k breaks are matched exactly.
- Algorithm idea: For each B_i, find exact matches of B_i in T
  Run [LV'89] only for positions in T where at least k breaks match exactly $\rightsquigarrow$ How?
- If $B_i = P[\ell_i \mathinner{.\,.}) = T[a \mathinner{.\,.})$, add a mark to $T[\lfloor (a - \ell_i)/k \rfloor]$; run [LV'89] for pos w/ $\geq k$ marks



Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ 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.
- Algorithm idea: For each B_i, find exact matches of B_i in T
  Run [LV'89] only for positions in T where at least k breaks match exactly $\rightsquigarrow$ How?
- If $B_i = P[\ell_i \mathinner{..}) = T[a \mathinner{..})$, add a mark to $T[\lfloor (a - \ell_i)/k \rfloor]$; run [LV'89] for pos w/ $\geq k$ marks

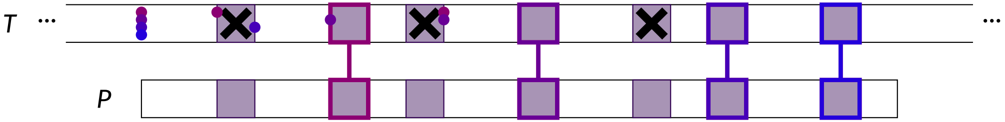


Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

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

- In any k-edit occ, at least k breaks are matched exactly.
- Algorithm idea: For each B_i, find exact matches of B_i in T

  Run [LV'89] only for positions in T where at least k breaks match exactly $\rightsquigarrow$ How?
- If $B_i = P[\ell_i \dots) = T[a \dots)$, add a mark to $T[\lfloor (a - \ell_i)/k \rfloor]$; run [LV'89] for pos w/ $\geq k$ marks

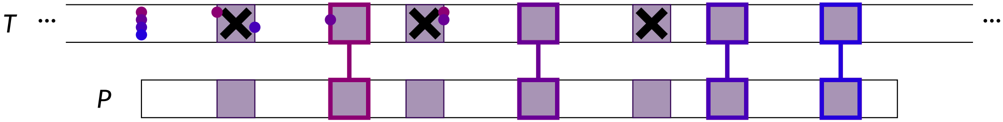


Have: $2k$ disjoint breaks $B_1, \dots, B_{2k}$ in P such that

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

- In any k-edit occ, at least k breaks are matched exactly.
- Algorithm idea: For each B_i, find exact matches of B_i in T

  Run [LV’89] only for positions in T where at least k breaks match exactly $\rightsquigarrow$ How?
- If $B_i = P[\,\ell_i \,..\,) = T[\,a\,..\,)$, add a mark to $T[\lfloor (a - \ell_i)/k \rfloor]$; run [LV’89] for pos w/ $\geq k$ marks $\rightsquigarrow O(k^2|T|/|P|)$ marks in total; $\rightsquigarrow O(k|T|/|P|)$ calls to [LV’89]; $\rightsquigarrow O(k^3|T|/|P|)$ time



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

Reduction to Periodic Patterns [CKoW'22]

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

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

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

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

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

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

[CKoW'22]

Reduction to Periodic Patterns

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

Need to tackle: P is almost periodic

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

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

[CKoW'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

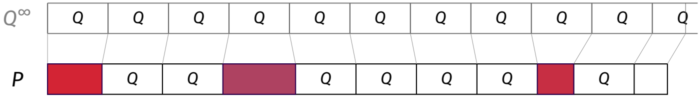
- In [CKoW'20], use elaborate marking scheme to obtain $O(|T| + |T|/|P| \cdot k^4)$ algorithm
  $\rightsquigarrow$ Not faster than [Cole,Hariharan'98]
- In [CKoW'22], trade-off between
  - Refinement of algorithm from [CKoW'20]
  - New algorithm based on "Seaweed Technology" of [Tiskin'10,'15]

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

Dynamic Puzzle Matching and How to Handle Not So Easy Patterns

Have: P is at ED $\leq 2k$ to a string with period Q of len $O(|P|/k)$

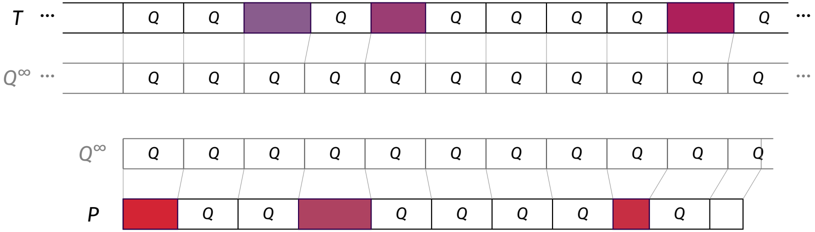
- If P is close to Q^∞, then so is T $\rightsquigarrow$ have alignments $A_P : P \rightsquigarrow\!\!\twoheadrightarrow Q^\infty$ and $A_T : T \rightsquigarrow\!\!\twoheadrightarrow Q^\infty$
  A_T and A_P induce tile partition of P and T; all but $O(k)$ tiles are Q



Have: P is at ED $\le 2k$ to a string with period Q of len $O(|P|/k)$

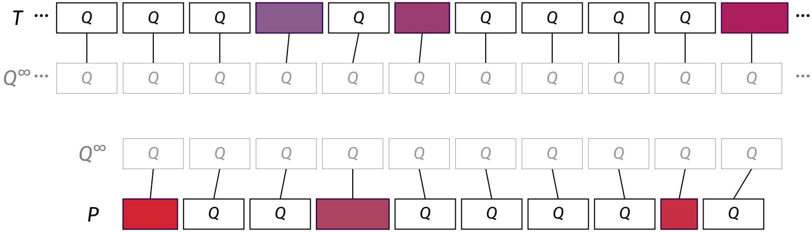
- If P is close to Q^∞, then so is $T \rightsquigarrow$ have alignments $A_P : P \rightsquigarrow\!\!\!\twoheadrightarrow Q^\infty$ and $A_T : T \rightsquigarrow\!\!\!\twoheadrightarrow Q^\infty$

  A_T and A_P induce tile partition of P and T; all but $O(k)$ tiles are Q



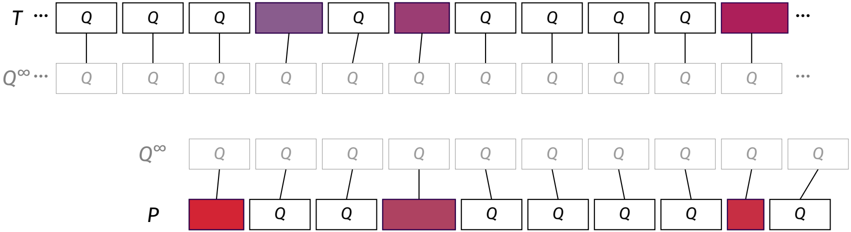
Have: P is at ED $\leq 2k$ to a string with period Q of len $O(|P|/k)$

- If P is close to Q^∞, then so is $T \rightsquigarrow$ have alignments $A_P : P \twoheadrightarrow Q^\infty$ and $A_T : T \twoheadrightarrow Q^\infty$
  A_T and A_P induce tile partition of P and T; all but $O(k)$ tiles are Q



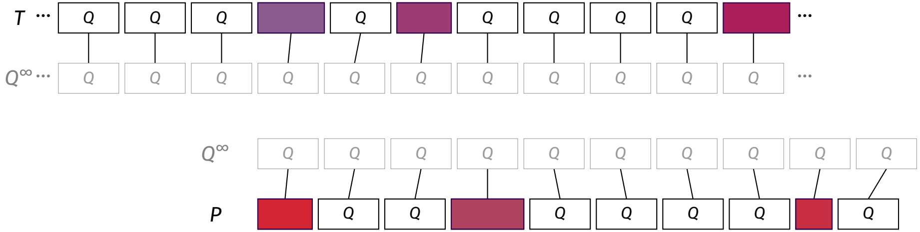
Have: P is at ED $\leq 2k$ to a string with period Q of len $O(|P|/k)$

- If P is close to Q^∞, then so is $T \rightsquigarrow$ have alignments $A_P : P \rightsquigarrow\!\!\!\!\twoheadrightarrow Q^\infty$ and $A_T : T \rightsquigarrow\!\!\!\!\twoheadrightarrow Q^\infty$
  A_T and A_P induce tile partition of P and T; all but $O(k)$ tiles are Q
- Imagine shifting P along T, one tile/Q at a time
  $\rightsquigarrow$ For each shift, want to compute occs
  $\rightsquigarrow$ Between shifts, $O(k)$ tiles get aligned to a new/different tile



Have: P is at ED $\leq 2k$ to a string with period Q of len $O(|P|/k)$

- If P is close to Q^∞, then so is $T \rightsquigarrow$ have alignments $A_P : P \rightsquigarrow\!\!\!\!\rightarrow Q^\infty$ and $A_T : T \rightsquigarrow\!\!\!\!\rightarrow Q^\infty$
  A_T and A_P induce tile partition of P and T; all but $O(k)$ tiles are Q
- Imagine shifting P along T, one tile/Q at a time
  $\rightsquigarrow$ For each shift, want to compute occs
  $\rightsquigarrow$ Between shifts, $O(k)$ tiles get aligned to a new/different tile



Have: P is at ED $\le 2k$ to a string with period Q of len $O(|P|/k)$

- If P is close to Q^∞, then so is $T \rightsquigarrow$ have alignments $A_P : P \rightsquigarrow\!\!\!\!\twoheadrightarrow Q^\infty$ and $A_T : T \rightsquigarrow\!\!\!\!\twoheadrightarrow Q^\infty$
  A_T and A_P induce tile partition of P and T; all but $O(k)$ tiles are Q
- Imagine shifting P along T, one tile/Q at a time
  $\rightsquigarrow$ For each shift, want to compute occs
  $\rightsquigarrow$ Between shifts, $O(k)$ tiles get aligned to a new/different tile

Dynamic Puzzle Matching [CKoW'22]

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2
Updates: Insertions and Deletions of pairs in I
Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Have: P is at ED $\leq 2k$ to a string with period Q of len $O(|P|/k)$

- If P is close to Q^∞, then so is T $\leadsto$ have alignments $A_P : P \twoheadrightarrow Q^\infty$ and $A_T : T \twoheadrightarrow Q^\infty$
  A_T and A_P induce tile partition of P and T; all but $O(k)$ tiles are Q
- Imagine shifting P along T, one tile/Q at a time
  $\leadsto$ For each shift, want to compute occs
  $\leadsto$ Between shifts, $O(k)$ tiles get aligned to a new/different tile

Dynamic Puzzle Matching [CKoW'22]

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2
Updates: Insertions and Deletions of pairs in I
Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result (using Seaweeds) [CKoW'22]

After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

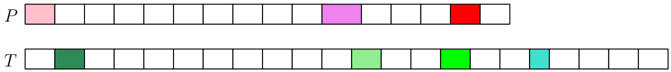
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs



[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

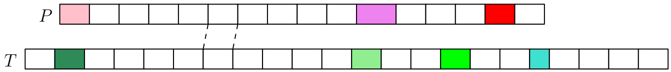
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs
- $> k$ copies of Q in $P \implies \geq 1$ copy of Q matched exactly



[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

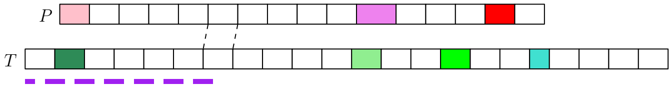
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs
- $> k$ copies of Q in $P \implies \geq 1$ copy of Q matched exactly
  $\rightsquigarrow$ Starting pos of k-edit occ's in T within $O(k)$ from endpoints of tiles



Dynamic Puzzle Matching [CKoW'22]

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F\in\mathrm{F}} \mathrm{ED}(F,Q) = O(k)$.

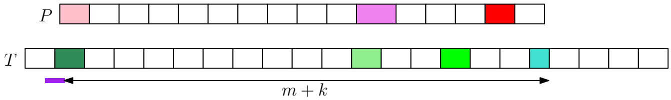
Maintain: sequence $I = (U_1,V_1)\cdots(U_z,V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1\cdots U_z$ in $V_1\cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs
- $> k$ copies of Q in $P \implies \geq 1$ copy of Q matched exactly
  $\rightsquigarrow$ Starting pos of k-edit occ's in T within $O(k)$ from endpoints of tiles



[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

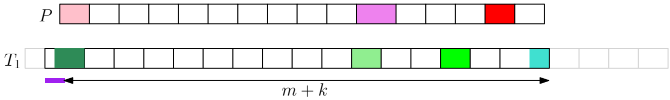
Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs
- $> k$ copies of Q in $P \implies \geq 1$ copy of Q matched exactly

  $\rightsquigarrow$ Starting pos of k-edit occ's in T within $O(k)$ from endpoints of tiles



[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

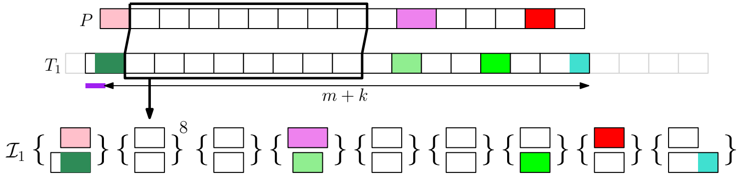
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs



[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in F} \mathrm{ED}(F, Q) = O(k)$.

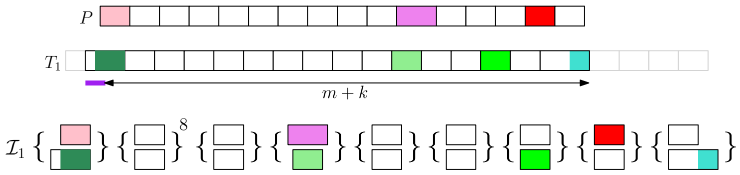
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs
- Goal: Iterate over all I_j's with one DPM instance



[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F\in F} \mathrm{ED}(F,Q) = O(k)$.

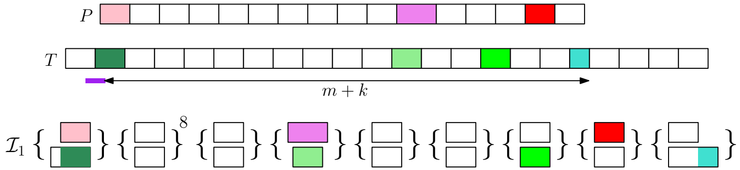
Maintain: sequence $I = (U_1, V_1)\cdots(U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs
- Goal: Iterate over all I_j's with one DPM instance



[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F\in F} \mathrm{ED}(F,Q) = O(k)$.

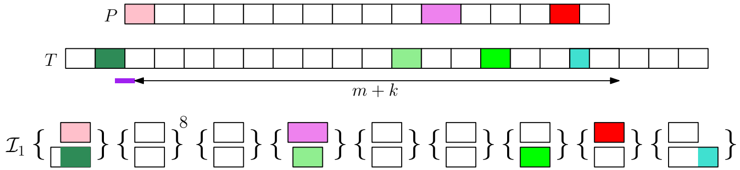
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs
- Goal: Iterate over all I_j's with one DPM instance



Dynamic Puzzle Matching

[CKoW'22]

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

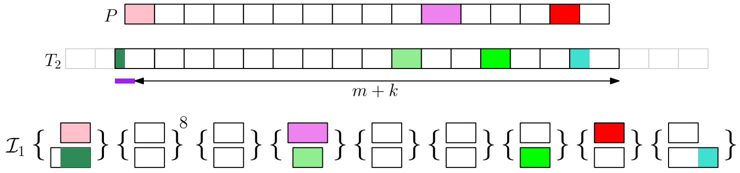
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs
- Goal: Iterate over all I_j's with one DPM instance



[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

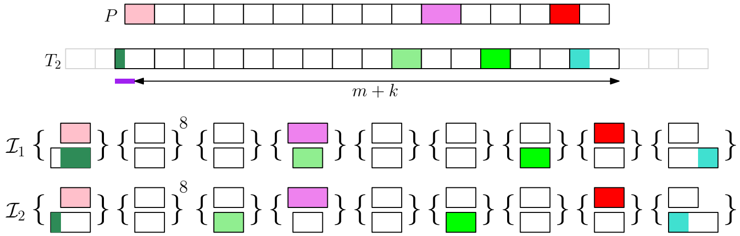
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs
- Goal: Iterate over all I_j's with one DPM instance



Dynamic Puzzle Matching [CKoW'22]

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F\in F} \mathrm{ED}(F,Q) = O(k)$.

Maintain: sequence $I = (U_1,V_1)\cdots(U_z,V_z)$ of pairs from F^2

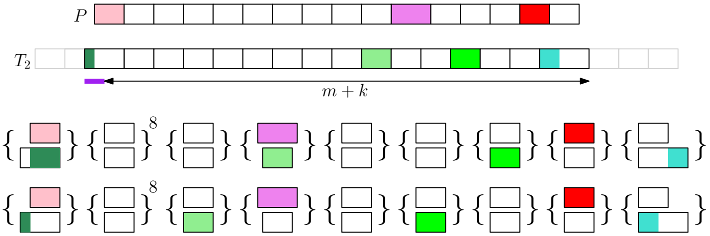
Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1\cdots U_z$ in $V_1\cdots V_z$

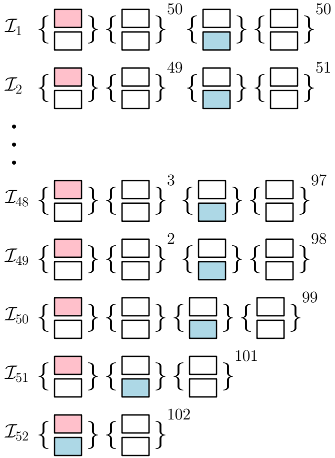
Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

- For simplicity: assume $|Q| \approx \sqrt{|P|}$; ignore handling of initial and final pairs
- Goal: Iterate over all I_j's with one DPM instance

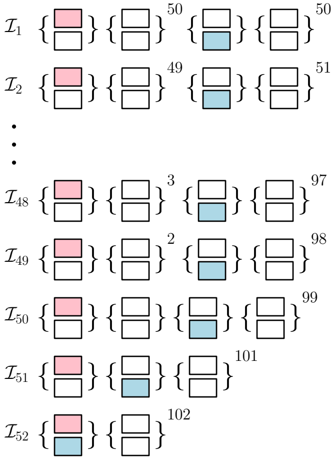
$\rightsquigarrow$ Over $\Theta(\sqrt{|P|})$ shifts of P, need $O(\sqrt{|P|}k)$ DPM-updates $\rightsquigarrow$ $\tilde{O}(k^3 + \sqrt{|P|}k^2)$ time



Example $k = 2$.


$\mathcal{I}_1$ 50 50
$\mathcal{I}_2$ 49 51
$\mathcal{I}_{48}$ 3 97
$\mathcal{I}_{49}$ 2 98
$\mathcal{I}_{50}$ 99
$\mathcal{I}_{51}$ 101
$\mathcal{I}_{52}$ 102

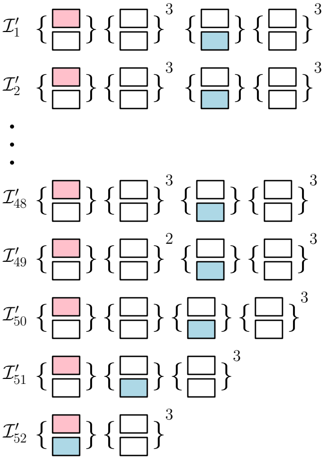
Example $k = 2$.



- For plain run $(Q, Q)^y$, at least $y - k$ copies of Q matched exactly in k-edit occ

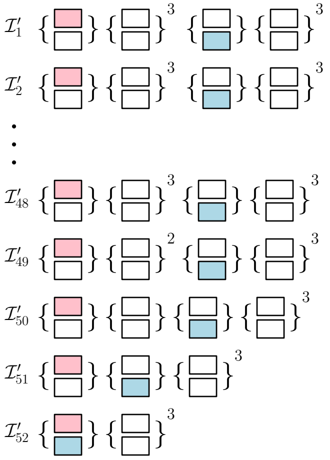
Using Dynamic Puzzle Matching *Better*

Example $k = 2$.



- For plain run $(Q, Q)^y$, at least $y - k$ copies of Q matched exactly in k-edit occ
- Cap exponents of plain runs at $k + 1$

Example $k = 2$.

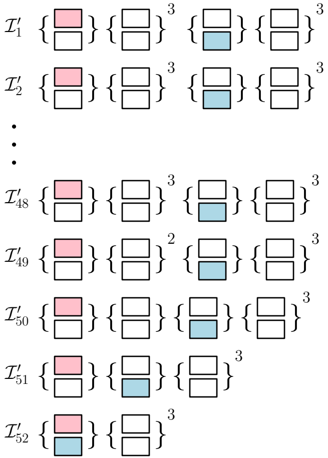


- For plain run $(Q, Q)^y$, at least $y - k$ copies of Q matched exactly in k-edit occ
- Cap exponents of plain runs at $k + 1$

  $\rightsquigarrow$ $O(k)$ DPM-updates per pair of special tiles

  $\rightsquigarrow$ $O(k^2)$ pairs of special tiles $\rightsquigarrow$ $\tilde{O}(k^4)$ time

Example $k = 2$.

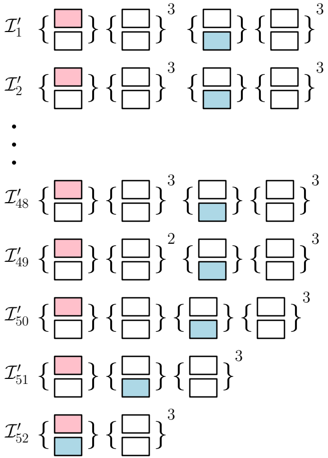


- For plain run $(Q, Q)^y$, at least $y - k$ copies of Q matched exactly in k-edit occ
- Cap exponents of plain runs at $\sqrt{k}$

  $\rightsquigarrow$ $O(\sqrt{k})$ DPM-updates per pair of special tiles

  $\rightsquigarrow$ $O(k^2)$ pairs of special tiles $\rightsquigarrow$ $\tilde{O}(k^{3.5})$ time

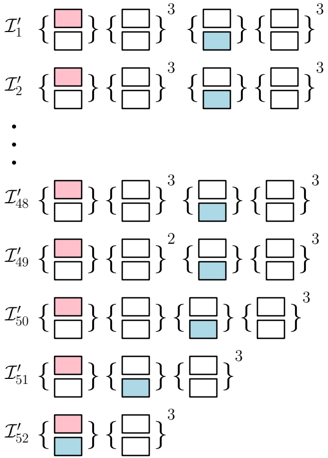
Example $k = 2$.



- For plain run $(Q, Q)^y$, at least $y - k$ copies of Q matched exactly in k-edit occ
- Cap exponents of plain runs at $\sqrt{k}$
  $\rightsquigarrow$ $O(\sqrt{k})$ DPM-updates per pair of special tiles
  $\rightsquigarrow$ $O(k^2)$ pairs of special tiles $\rightsquigarrow$ $\tilde{O}(k^{3.5})$ time
- False positives if $\geq \sqrt{k}$ edits in run of (Q, Q)!

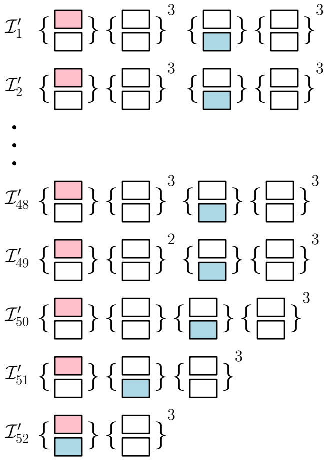
Using Dynamic Puzzle Matching *Better*

Example $k = 2$.



- For plain run $(Q, Q)^y$, at least $y - k$ copies of Q matched exactly in k-edit occ
- Cap exponents of plain runs at $\sqrt{k}$
  $\rightsquigarrow$ $O(\sqrt{k})$ DPM-updates per pair of special tiles
  $\rightsquigarrow$ $O(k^2)$ pairs of special tiles $\rightsquigarrow$ $\tilde{O}(k^{3.5})$ time
- False positives if $\geq \sqrt{k}$ edits in run of (Q, Q)!
  $\rightsquigarrow$ Filter out false positives using another marking scheme $\rightsquigarrow$ $\tilde{O}(k^{3.5})$ time in total (boring)

Example $k = 2$.



- For plain run $(Q, Q)^y$, at least $y - k$ copies of Q matched exactly in k-edit occ
- Cap exponents of plain runs at $\sqrt{k}$

  $\rightsquigarrow O(\sqrt{k})$ DPM-updates per pair of special tiles

  $\rightsquigarrow O(k^2)$ pairs of special tiles $\rightsquigarrow \tilde{O}(k^{3.5})$ time
- False positives if $\geq \sqrt{k}$ edits in run of (Q, Q)!

  $\rightsquigarrow$ Filter out false positives using another marking scheme $\rightsquigarrow \tilde{O}(k^{3.5})$ time in total (boring)

Main Result [CKoW'22]

Pattern P, text T, threshold k; can compute starting pos of all k-edit occ's in time $\tilde{O}(|T| + k^{3.5}\ |T|/|P|)$.

[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Goal: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

General Idea

1. Store edit distance information for each pair (U_i, V_i) $\rightsquigarrow$ suitable permutation matrices allow this in $O(k)$ space
2. Show how to compose the information of different pairs in a suitable way $\rightsquigarrow$ "seaweed product"
3. Show how to compute said product efficiently

[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Goal: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

General Idea

1. Store edit distance information for each pair $(U_i, V_i) \rightsquigarrow$ suitable permutation matrices allow this in $O(k)$ space
2. Show how to compose the information of different pairs in a suitable way $\rightsquigarrow$ "seaweed product"
3. Show how to compute said product efficiently

[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Goal: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

General Idea

1. Store edit distance information for each pair $(U_i, V_i) \rightsquigarrow$ suitable permutation matrices allow this in $O(k)$ space
2. Show how to compose the information of different pairs in a suitable way $\rightsquigarrow$ "seaweed product"
3. Show how to compute said product efficiently

[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathsf{F}} \mathrm{ED}(F, Q) = O(k)$.

Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Goal: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

General Idea

1. Store edit distance information for each pair (U_i, V_i) $\rightsquigarrow$ suitable permutation matrices allow this in $O(k)$ space
2. Show how to compose the information of different pairs in a suitable way $\rightsquigarrow$ "seaweed product"
3. Show how to compute said product efficiently

[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F \in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.
Maintain: sequence $I = (U_1, V_1) \cdots (U_z, V_z)$ of pairs from F^2
Updates: Insertions and Deletions of pairs in I
Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Goal: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

General Idea

1. Store edit distance information for each pair (U_i, V_i) $\rightsquigarrow$ suitable permutation matrices allow this in $O(k)$ space
2. Show how to compose the information of different pairs in a suitable way $\rightsquigarrow$ "seaweed product"
3. Show how to compute said product efficiently

Content Warning: Some technical computations ahead!

- $A \in \{0,1\}^{n \times n}$ is permutation matrix: every row and every column has exactly one 1
  $\rightsquigarrow$ 1-to-1 corresponds to permutation in S_n
- For $n \times n$ matrix A, distribution matrix A^{Σ} is $(n+1) \times (n+1)$ matrix with

$$A^{\Sigma}[\,i,j\,] := \sum_{i' \geq i} \sum_{j' < j} A[\,i',j'\,]$$

- For $(n+1) \times (n+1)$ matrix A, density matrix $A^{\square}$ is $n \times n$ matrix with

$$A^{\square}[\,i,j\,] := A[\,i+1,j\,] + A[\,i,j+1\,] - A[\,i+1,j+1\,] - A[\,i,j\,]$$

$$\begin{array}{ccc} (0,0) & & (n-1,0) \\ & \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} & \\ (0,n-1) & & (n-1,n-1) \end{array}$$

- $A \in \{0,1\}^{n \times n}$ is **permutation matrix**: every row and every column has exactly one 1
  $\rightsquigarrow$ 1-to-1 corresponds to permutation in S_n
- For $n \times n$ matrix A, **distribution matrix** A^{Σ} is $(n+1) \times (n+1)$ matrix with

$$A^{\Sigma}[\,i, j\,] := \sum_{i' \geq i} \sum_{j' < j} A[\,i', j'\,]$$

- For $(n+1) \times (n+1)$ matrix A, **density matrix** $A^{\square}$ is $n \times n$ matrix with

$$A^{\square}[\,i, j\,] := A[\,i+1, j\,] + A[\,i, j+1\,] - A[\,i+1, j+1\,] - A[\,i, j\,]$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}$$

- $A \in \{0,1\}^{n \times n}$ is permutation matrix: every row and every column has exactly one 1
  $\rightsquigarrow$ 1-to-1 corresponds to permutation in S_n
- For $n \times n$ matrix A, distribution matrix A^Σ is $(n+1) \times (n+1)$ matrix with

$$A^\Sigma[i,j] := \sum_{i' \geq i} \sum_{j' < j} A[i',j']$$

- For $(n+1) \times (n+1)$ matrix A, density matrix $A^\square$ is $n \times n$ matrix with

$$A^\square[i,j] := A[i+1,j] + A[i,j+1] - A[i+1,j+1] - A[i,j]$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}^\Sigma = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & \boxed{1} & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

- $A \in \{0,1\}^{n \times n}$ is permutation matrix: every row and every column has exactly one 1
  $\rightsquigarrow$ 1-to-1 corresponds to permutation in S_n
- For $n \times n$ matrix A, distribution matrix A^{Σ} is $(n+1) \times (n+1)$ matrix with

$$A^{\Sigma}[\,i,j\,] := \sum_{i' \geq i} \sum_{j' < j} A[\,i',j'\,]$$

- For $(n+1) \times (n+1)$ matrix A, density matrix $A^{\square}$ is $n \times n$ matrix with

$$A^{\square}[\,i,j\,] := A[\,i+1,j\,] + A[\,i,j+1\,] - A[\,i+1,j+1\,] - A[\,i,j\,]$$

$$\begin{pmatrix} 0 & \boxed{1} & \boxed{0} \\ 1 & \boxed{0} & \boxed{0} \\ 0 & 0 & 1 \end{pmatrix}^{\Sigma} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & \boxed{1} & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}^{\square} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & \boxed{0} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- $A \in \{0,1\}^{n \times n}$ is permutation matrix: every row and every column has exactly one 1
  $\rightsquigarrow$ 1-to-1 corresponds to permutation in S_n
- For $n \times n$ matrix A, distribution matrix A^{Σ} is $(n+1) \times (n+1)$ matrix with

$$A^{\Sigma}[\,i,j\,] := \sum_{i' \geq i} \sum_{j' < j} A[\,i',j'\,]$$

- For $(n+1) \times (n+1)$ matrix A, density matrix $A^{\square}$ is $n \times n$ matrix with

$$A^{\square}[\,i,j\,] := A[\,i+1,j\,] + A[\,i,j+1\,] - A[\,i+1,j+1\,] - A[\,i,j\,]$$

Lemma (Relation Density-Distribution Matrix)

For any A, have $(A^{\Sigma})^{\square} = A$; for simple A, have $(A^{\square})^{\Sigma} = A$ (first row and last col of A are 0)

- $A \in \{0,1\}^{n \times n}$ is permutation matrix: every row and every column has exactly one 1
  $\rightsquigarrow$ 1-to-1 corresponds to permutation in S_n
- For $n \times n$ matrix A, distribution matrix A^{Σ} is $(n+1) \times (n+1)$ matrix with

$$A^{\Sigma}[i,j] := \sum_{i' \geq i} \sum_{j' < j} A[i',j']$$

- For $(n+1) \times (n+1)$ matrix A, density matrix $A^{\square}$ is $n \times n$ matrix with

$$A^{\square}[i,j] := A[i+1,j] + A[i,j+1] - A[i+1,j+1] - A[i,j]$$

Lemma (Relation Density-Distribution Matrix)

For any A, have $(A^{\Sigma})^{\square} = A$; for simple A, have $(A^{\square})^{\Sigma} = A$ (first row and last col of A are 0)

- $A \in \{0,1\}^{n\times n}$ is permutation matrix: every row and every column has exactly one 1
  $\rightsquigarrow$ 1-to-1 corresponds to permutation in S_n
- For $n \times n$ matrix A, distribution matrix A^Σ is $(n+1)\times(n+1)$ matrix with

$$A^\Sigma[i,j] := \sum_{i' \geq i} \sum_{j' < j} A[i',j']$$

- For $(n+1)\times(n+1)$ matrix A, density matrix $A^\square$ is $n \times n$ matrix with

$$A^\square[i,j] := A[i+1,j] + A[i,j+1] - A[i+1,j+1] - A[i,j]$$

Lemma (Relation Density-Distribution Matrix)

For any A, have $(A^\Sigma)^\square = A$; for simple A, have $(A^\square)^\Sigma = A$ (first row and last col of A are 0)

Lemma (Properties of Permutation Matrices)

Perm. matrix A; $A^\Sigma[i,0] = A^\Sigma[n-1,j] = 0$ and $A^\Sigma[0,j] = j$; $A^\Sigma[i,0] = n-1-i$ and $A^\Sigma[i,j] \geq j-i$

- $A \in \{0,1\}^{n \times n}$ is permutation matrix: every row and every column has exactly one 1
- For $n \times n$ matrix A, distribution matrix A^{Σ} is $(n+1) \times (n+1)$ matrix with

$$A^{\Sigma}[\,i,j\,] := \sum_{i' \geq i} \sum_{j' < j} A[\,i',j'\,]$$

- For $(n+1) \times (n+1)$ matrix A, density matrix $A^{\square}$ is $n \times n$ matrix with

$$A^{\square}[\,i,j\,] := A[\,i+1,j\,] + A[\,i,j+1\,] - A[\,i+1,j+1\,] - A[\,i,j\,]$$

- A is simple: first row and last col are 0; for simple A have $A^{\Sigma\square} = A^{\square\Sigma} = A$
- A is Monge if $A^{\square} \geq 0$; A is unit-Monge if $A^{\square}$ is a permutation matrix

- $A \in \{0,1\}^{n \times n}$ is permutation matrix: every row and every column has exactly one 1
- For $n \times n$ matrix A, distribution matrix A^{Σ} is $(n+1) \times (n+1)$ matrix with

$$A^{\Sigma}[\,i,j\,] := \sum_{i' \geq i} \sum_{j' < j} A[\,i',j'\,]$$

- For $(n+1) \times (n+1)$ matrix A, density matrix $A^{\square}$ is $n \times n$ matrix with

$$A^{\square}[\,i,j\,] := A[\,i+1,j\,] + A[\,i,j+1\,] - A[\,i+1,j+1\,] - A[\,i,j\,]$$

- A is simple: first row and last col are 0; for simple A have $A^{\Sigma\square} = A^{\square\Sigma} = A$
- A is Monge if $A^{\square} \geq 0$; A is unit-Monge if $A^{\square}$ is a permutation matrix

Lemma (Working with Permutation Matrices) [Tis07, CP10]

For $n \times n$ permutation matrix A
can store in $O(n)$ space and can access $A^{\Sigma}[\,i,j\,]$ in $O(\log n / \log\log n)$

- $A \in \{0,1\}^{n \times n}$ is permutation matrix: every row and every column has exactly one 1
- For $n \times n$ matrix A, distribution matrix A^{Σ} is $(n+1) \times (n+1)$ matrix with

$$A^{\Sigma}[i,j] := \sum_{i' \geq i} \sum_{j' < j} A[i',j']$$

- For $(n+1) \times (n+1)$ matrix A, density matrix $A^{\square}$ is $n \times n$ matrix with

$$A^{\square}[i,j] := A[i+1,j] + A[i,j+1] - A[i+1,j+1] - A[i,j]$$

- A is simple: first row and last col are 0;   for simple A have $A^{\Sigma\square} = A^{\square\Sigma} = A$
- A is Monge if $A^{\square} \geq 0$; A is unit-Monge if $A^{\square}$ is a permutation matrix

Lemma (Working with Permutation Matrices) [Tis07, CP10]

For $n \times n$ permutation matrix A

can store in $O(n)$ space   and   can access $A^{\Sigma}[i,j]$ in $O(\log n / \log\log n)$ (recall yesterday's Lem. 17)

Detour: Characterizing Unit-Monge Matrices

Lemma

Any matrix $A \in \mathbb{Z}_{\geq 0}^{n+1 \times n+1}$ with

- $A[\,i, 0\,] = 0$
- $A[\,n, j\,] = 0$
- $A[\,i, n\,] = n - i$
- $A[\,0, j\,] = j$

$$A \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

is a unit Monge matrix.

- Show: each row/column of $A^{\square}$ sums to 1
- Consider

$$\sum_j A^{\square}[\,i,j\,] = \sum_j A[\,i+1, j\,] + A[\,i, j+1\,] - A[\,i+1, j+1\,] - A[\,i,j\,]$$

$$= A[\,i+1, 0\,] + A[\,i, n\,] - A[\,i+1, n\,] - A[\,i, 0\,]$$

$$= 0 + (n-i) - (n-i-1) - 0$$

$$= 1$$

$$A \stackrel{!}{=} \left(\begin{array}{cc|c|c|cc} 0 & \cdots & 0 & 0 & \cdots & 0 \\ 1 & & & & & 0 \\ \cdots & & & & & \cdots \\ n-j & & - & + & & 0 \\ n-j+1 & & + & - & & 0 \\ \cdots & & & & & \cdots \\ n-1 & & & & & 0 \\ n & \cdots & n-i & n-i-1 & \cdots & 0 \end{array}\right)$$

Lemma

Any matrix $A \in \mathbb{Z}_{\geq 0}^{n+1 \times n+1}$ with

- $A[\,i, 0\,] = 0$
- $A[\,n, j\,] = 0$
- $A[\,i, n\,] = n - i$
- $A[\,0, j\,] = j$

$$A \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \vdots & & & & \vdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

is a unit Monge matrix.

- Show: each row/column of $A^{\square}$ sums to 1
- Consider

$$\sum_j A^{\square}[\,i, j\,] = \sum_j A[\,i+1, j\,] + A[\,i, j+1\,] - A[\,i+1, j+1\,] - A[\,i, j\,]$$

$$= A[\,i+1, 0\,] + A[\,i, n\,] - A[\,i+1, n\,] - A[\,i, 0\,]$$

$$= 0 + (n - i) - (n - i - 1) - 0$$

$$= 1$$

$$A \stackrel{!}{=} \left(\begin{array}{cc|c|c|cc} 0 & \cdots & 0^{-} & 0^{+} & \cdots & 0 \\ 1 & & +- & -+ & & 0 \\ \vdots & & +- & -+ & & \vdots \\ n-j & & +- & -+ & & 0 \\ n-j+1 & & +- & -+ & & 0 \\ \vdots & & +- & -+ & & \vdots \\ n-1 & & +- & -+ & & 0 \\ n & \cdots & n-i^{+} & n-i-1^{-} & \cdots & 0 \end{array}\right)$$

Lemma

Any matrix $A \in \mathbb{Z}_{\geq 0}^{n+1 \times n+1}$ with

- $A[\,i, 0\,] = 0$
- $A[\,n, j\,] = 0$
- $A[\,i, n\,] = n - i$
- $A[\,0, j\,] = j$

$$A \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

is a unit Monge matrix.

- Show: each row/column of $A^{\square}$ sums to 1
- Consider

$$\sum_j A^{\square}[\,i,j\,] = \sum_j A[\,i+1,j\,] + A[\,i,j+1\,] - A[\,i+1,j+1\,] - A[\,i,j\,]$$
$$= A[\,i+1,0\,] + A[\,i,n\,] - A[\,i+1,n\,] - A[\,i,0\,]$$
$$= 0 + (n-i) - (n-i-1) - 0$$
$$= 1$$

$$A \stackrel{!}{=} \left(\begin{array}{cc|c|c|cc} 0 & \cdots & 0^{-} & 0^{+} & \cdots & 0 \\ 1 & & +- & -+ & & 0 \\ \cdots & & +- & -+ & & \cdots \\ n-j & & +- & -+ & & 0 \\ n-j+1 & & +- & -+ & & 0 \\ \cdots & & +- & -+ & & \cdots \\ n-1 & & +- & -+ & & 0 \\ n & \cdots & n-i^{+} & n-i-1^{-} & \cdots & 0 \end{array}\right)$$

- A is simple: first row and last col are 0; for simple A have $A^{\Sigma\square} = A^{\square\Sigma} = A$
- A is Monge if $A^{\square} \geq 0$; A is unit-Monge if $A^{\square}$ is a permutation matrix
- For matrices A, B, the (min-+)-product $A \odot B$ is

$$(A \odot B)[i,k] := \min_{j}(A[i,j] + B[j,k])$$

Recall: For adjacency matrix A of (weighted) graph G, $A^{\odot \ell}$ stores ℓ-hop dist in G

- A is simple: first row and last col are 0; for simple A have $A^{\Sigma\square} = A^{\square\Sigma} = A$
- A is Monge if $A^{\square} \geq 0$; A is unit-Monge if $A^{\square}$ is a permutation matrix
- For matrices A, B, the (min-+)-product $A \odot B$ is

$$(A \odot B)[\,i,k\,] := \min_j (A[\,i,j\,] + B[\,j,k\,])$$

Recall: For adjacency matrix A of (weighted) graph G, $A^{\odot \ell}$ stores ℓ-hop dist in G

- A is simple: first row and last col are 0; for simple A have $A^{\Sigma\square} = A^{\square\Sigma} = A$
- A is Monge if $A^{\square} \geq 0$; A is unit-Monge if $A^{\square}$ is a permutation matrix
- For matrices A, B, the (min-+)-product $A \odot B$ is

$$(A \odot B)[i,k] := \min_{j}(A[i,j] + B[j,k])$$

Recall: For adjacency matrix A of (weighted) graph G, $A^{\odot \ell}$ stores ℓ-hop dist in G

- A is simple: first row and last col are 0; for simple A have $A^{\Sigma\square} = A^{\square\Sigma} = A$
- A is Monge if $A^{\square} \geq 0$; A is unit-Monge if $A^{\square}$ is a permutation matrix
- For matrices A, B, the (min-+)-product $A \odot B$ is

$$(A \odot B)[i,k] := \min_{j}(A[i,j] + B[j,k])$$

Recall: For adjacency matrix A of (weighted) graph G, $A^{\odot\ell}$ stores ℓ-hop dist in G

[Tis'07,'15]

Lemma (Unit-Monge Monoid)

For Monge A, B, the matrix $A \odot B$ is Monge.

For simple unit-Monge A, B, the matrix $A \odot B$ is simple unit-Monge.

- A is simple: first row and last col are 0; for simple A have $A^{\Sigma\square} = A^{\square\Sigma} = A$
- A is Monge if $A^{\square} \geq 0$; A is unit-Monge if $A^{\square}$ is a permutation matrix
- For matrices A, B, the (min-+)-product $A \odot B$ is

$$(A \odot B)[i,k] := \min_j (A[i,j] + B[j,k])$$

Recall: For adjacency matrix A of (weighted) graph G, $A^{\odot \ell}$ stores ℓ-hop dist in G

[Tis'07,'15]

Lemma (Unit-Monge Monoid)

For Monge A, B, the matrix $A \odot B$ is Monge.
For simple unit-Monge A, B, the matrix $A \odot B$ is simple unit-Monge.

- A is simple: first row and last col are 0; for simple A have $A^{\Sigma\Box} = A^{\Box\Sigma} = A$
- A is Monge if $A^{\Box} \geq 0$; A is unit-Monge if $A^{\Box}$ is a permutation matrix
- For matrices A, B, the (min-+)-product $A \odot B$ is

$$(A \odot B)[\,i,k\,] := \min_j (A[\,i,j\,] + B[\,j,k\,])$$

Recall: For adjacency matrix A of (weighted) graph G, $A^{\odot\ell}$ stores ℓ-hop dist in G

[Tis'07,'15]

Lemma (Unit-Monge Monoid)

For Monge A, B, the matrix $A \odot B$ is Monge.
For simple unit-Monge A, B, the matrix $A \odot B$ is simple unit-Monge.

Proof: Board or exercise.

Monge Matrix × (min,+)-Product

- A is simple: first row and last col are 0; for simple A have $A^{\Sigma\square} = A^{\square\Sigma} = A$
- A is Monge if $A^{\square} \geq 0$; A is unit-Monge if $A^{\square}$ is a permutation matrix
- For matrices A, B, the (min-+)-product $A \odot B$ is

$$(A \odot B)[i,k] := \min_j (A[i,j] + B[j,k])$$

[Tis'07,'15]

Lemma (Unit-Monge Monoid)

For Monge A, B, the matrix $A \odot B$ is Monge.
For simple unit-Monge A, B, the matrix $A \odot B$ is simple unit-Monge.

- For perm matrices A, B, define seaweed product $A \boxdot B := (A^{\Sigma} \odot B^{\Sigma})^{\square}$
- $A \boxdot B$ is different from normal perm concat: $I^R = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ \vdots & 0 & 1 & 0 \\ 0 & 1 & 0 & \vdots \\ 1 & 0 & \cdots & 0 \end{pmatrix}$ is a zero

- A is simple: first row and last col are 0; for simple A have $A^{\Sigma\square} = A^{\square\Sigma} = A$
- A is Monge if $A^{\square} \geq 0$; A is unit-Monge if $A^{\square}$ is a permutation matrix
- For matrices A, B, the (min-+)-product $A \odot B$ is

$$(A \odot B)[i,k] := \min_j (A[i,j] + B[j,k])$$

[Tis'07,'15]

Lemma (Unit-Monge Monoid)

For Monge A, B, the matrix $A \odot B$ is Monge.
For simple unit-Monge A, B, the matrix $A \odot B$ is simple unit-Monge.

- For perm matrices A, B, define seaweed product $A \boxdot B := (A^{\Sigma} \odot B^{\Sigma})^{\square}$
- $A \boxdot B$ is different from normal perm concat: $I^R = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ | & 0 & 1 & 0 \\ 0 & 1 & 0 & | \\ 1 & 0 & \cdots & 0 \end{pmatrix}$ is a zero

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} =$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} =$$

$$\big\downarrow (\cdot)^{\Sigma} \qquad \big\downarrow (\cdot)^{\Sigma}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} =$$

$$\rightsquigarrow (\cdot)^{\Sigma} \qquad \rightsquigarrow (\cdot)^{\Sigma}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\rightsquigarrow (\cdot)^{\Sigma} \qquad \rightsquigarrow (\cdot)^{\Sigma} \qquad \leftsquigarrow (\cdot)^{\square}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} =$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} =$$

$$\Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\downarrow (\cdot)^{\Sigma}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} =$$

$$\Big\downarrow (\cdot)^{\Sigma} \qquad \Big\downarrow (\cdot)^{\Sigma}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$\rightsquigarrow (\cdot)^{\Sigma} \qquad \rightsquigarrow (\cdot)^{\Sigma} \qquad \leftsquigarrow (\cdot)^{\square}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

Naive computation: $O(n^3) \rightsquigarrow$ too slow

Naive computation: $O(n^3) \rightsquigarrow$ too slow

Theorem (Monge Multiplication) [SMAWK'87]

Can compute (min,+)-product of two $n \times n$ Monge matrices in time $\tilde{O}(n^2)$.

Naive computation: $O(n^3) \rightsquigarrow$ too slow

Theorem (Monge Multiplication) [SMAWK'87]

Can compute (min,+)-product of two $n \times n$ Monge matrices in time $\tilde{O}(n^2)$.

(Proof skipped.)

Naive computation: $O(n^3) \rightsquigarrow$ too slow

Theorem (Monge Multiplication) [SMAWK'87]

Can compute (min,+)-product of two $n \times n$ Monge matrices in time $\tilde{O}(n^2)$.

(Proof skipped.)

Theorem (Unit Monge Multiplication) [Tis'07,'15]

Can compute (min,+)-product of two $n \times n$ simple unit-Monge matrices in time $O(n \log n)$.

Naive computation: $O(n^3) \rightsquigarrow$ too slow

Theorem (Monge Multiplication) [SMAWK'87]

Can compute (min,+)-product of two $n \times n$ Monge matrices in time $\tilde{O}(n^2)$.

(Proof skipped.)

Theorem (Unit Monge Multiplication) [Tis'07,'15]

Can compute (min,+)-product of two $n \times n$ simple unit-Monge matrices in time $O(n \log n)$.

Proof idea (details messy, but not too complicated).

- Divide and Conquer: Split each matrix into two; reduce to just two subproblems
- For conquer step, use elaborate, but easy to compute function to decide which solution to propagate from subproblem

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\rightsquigarrow (\cdot)^{\Sigma} \qquad \rightsquigarrow (\cdot)^{\Sigma} \qquad \leftsquigarrow (\cdot)^{\square}$$

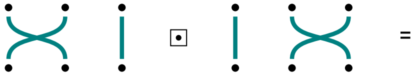
$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

Monge Matrix × (min,+)-Product—Examples, revisited

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\uparrow (\cdot)^{\square}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

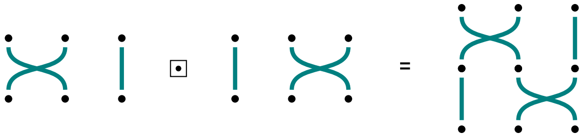


Monge Matrix × (min,+)-Product—Examples, revisited

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\downarrow (\cdot)^{\Sigma} \qquad \downarrow (\cdot)^{\Sigma} \qquad \uparrow (\cdot)^{\square}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

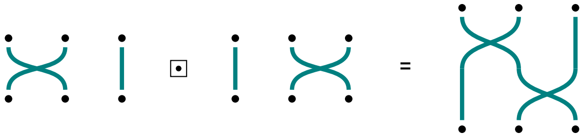


Monge Matrix × (min,+)-Product—Examples, revisited

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\downarrow (\cdot)^{\Sigma} \qquad\qquad \downarrow (\cdot)^{\Sigma} \qquad\qquad \uparrow (\cdot)^{\square}$$

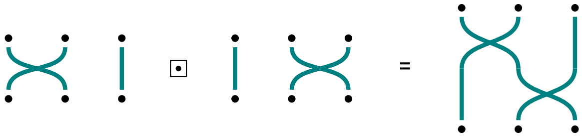
$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$



$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\uparrow (\cdot)^{\square}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$



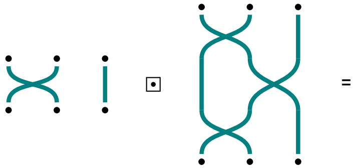
Example matches normal multiplication of permutation matrices.

$$
\begin{array}{ccccc}
\begin{pmatrix}0&1&0\\1&0&0\\0&0&1\end{pmatrix} & \boxdot & \begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix} & = & \begin{pmatrix}0&0&1\\0&1&0\\1&0&0\end{pmatrix} \\
\rightsquigarrow \ (\cdot)^{\Sigma} & & \rightsquigarrow \ (\cdot)^{\Sigma} & & \leftsquigarrow \ (\cdot)^{\square} \\
\begin{pmatrix}0&0&0&0\\1&1&0&0\\2&1&0&0\\3&2&1&0\end{pmatrix} & \odot & \begin{pmatrix}0&0&0&0\\1&1&1&0\\2&2&1&0\\3&2&1&0\end{pmatrix} & = & \begin{pmatrix}0&0&0&0\\1&1&1&0\\2&2&1&0\\3&2&1&0\end{pmatrix}
\end{array}
$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$\Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\uparrow (\cdot)^{\square}$$

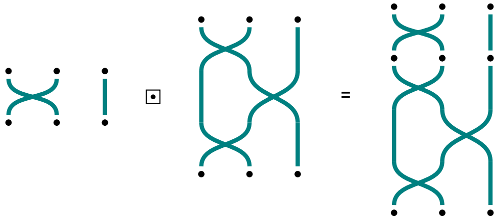
$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$



$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$\Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\uparrow (\cdot)^{\square}$$

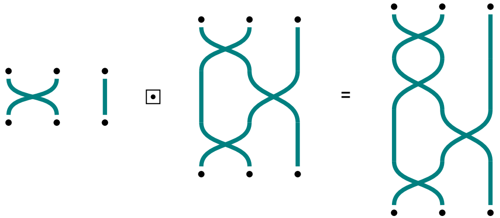
$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$



$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$\downarrow (\cdot)^{\Sigma} \qquad \downarrow (\cdot)^{\Sigma} \qquad \uparrow (\cdot)^{\square}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

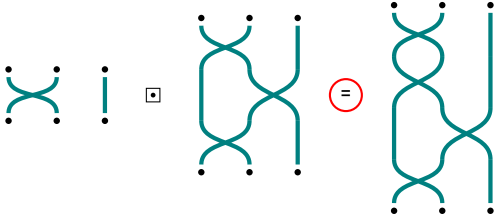


Monge Matrix × (min,+)-Product—Examples, revisited

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$\Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\downarrow (\cdot)^{\Sigma} \qquad\qquad \Big\uparrow (\cdot)^{\square}$$

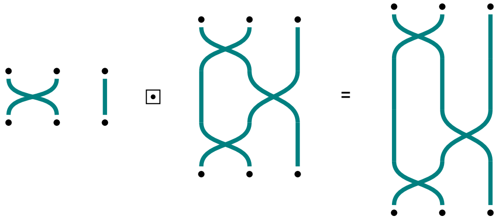
$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$



$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$\rightsquigarrow (\cdot)^{\Sigma} \qquad \rightsquigarrow (\cdot)^{\Sigma} \qquad \leftsquigarrow (\cdot)^{\square}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$

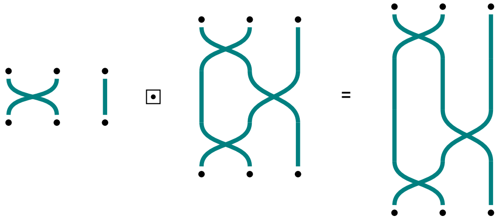


Monge Matrix × (min,+)-Product—Examples, revisited

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \boxdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

$$\downarrow (\cdot)^{\Sigma} \qquad\qquad \downarrow (\cdot)^{\Sigma} \qquad\qquad \uparrow (\cdot)^{\square}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 2 & 2 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{pmatrix}$$



Can view seaweed product in terms of these diagrams, but need some extra simplification rules.

Tis'07,'15

Seaweed Monoid

Can characterize monoid (simple unit-Monge, ⊡) in terms of seaweed diagrams with their stitching, when using as generators the elementary seaweeds

and the following relations

(idempotence for every generator)

(far commutativity)

(braid relation)

- Elementary seaweeds are just identity matrix and transposition matrices.
- Idempotence is crucial difference from standard perm concat

Tis'07,'15

Seaweed Monoid

Can characterize monoid (simple unit-Monge, $\boxdot$) in terms of seaweed diagrams with their stitching, when using as generators the elementary seaweeds

and the following relations

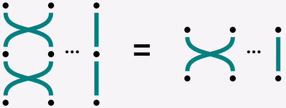
- Elementary seaweeds are just identity matrix and transposition matrices.
- Idempotence is crucial difference from standard perm concat

Tis'07,'15

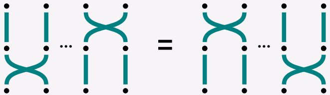
Seaweed Monoid

Can characterize monoid (simple unit-Monge, ⊡) in terms of seaweed diagrams with their stitching, when using as generators the elementary seaweeds

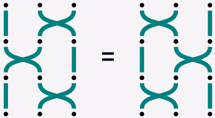
and the following relations



(idempotence for every generator)



(far commutativity)



(braid relation)

- Elementary seaweeds are just identity matrix and transposition matrices.
- Idempotence is crucial difference from standard perm concat

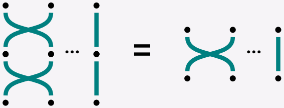
Monge Matrix × (min,+)-Product—Examples, revisited

Tis'07,'15

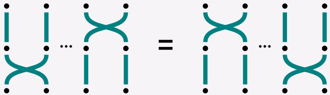
Seaweed Monoid

Can characterize monoid (simple unit-Monge, $\boxdot$) in terms of seaweed diagrams with their stitching, when using as generators the elementary seaweeds

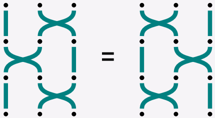
and the following relations



(idempotence for every generator)



(far commutativity)



(braid relation)

- Elementary seaweeds are just identity matrix and transposition matrices.
- Idempotence is crucial difference from standard perm concat

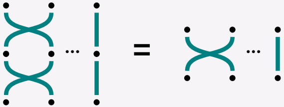
Monge Matrix × (min,+)-Product—Examples, revisited

Tis'07,'15

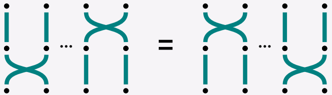
Seaweed Monoid

Can characterize monoid (simple unit-Monge, $\boxdot$) in terms of seaweed diagrams with their stitching, when using as generators the elementary seaweeds

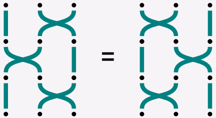
and the following relations



(idempotence for every generator)



(far commutativity)



(braid relation)

- Elementary seaweeds are just identity matrix and transposition matrices.
- Idempotence is crucial difference from standard perm concat

- Consider following task:
  Have permutations $(1, \dots, n) \mapsto (\sigma(1), \dots, \sigma(n))$ and $(1, \dots, n) \mapsto (\varrho(1), \dots, \varrho(n))$
  Want permutation $(1, \dots, n, n+1, \dots, 2n) \mapsto (\sigma(1), \dots, \sigma(n), \varrho(1), \dots, \varrho(n))$
  $\rightsquigarrow$ Prime application: string concatenation
- Claim:

$$P_{\sigma\varrho} = \begin{pmatrix} P_\sigma & 0 \\ 0 & P_\varrho \end{pmatrix} = \begin{pmatrix} P_\sigma & 0 \\ 0 & I_{|\varrho|} \end{pmatrix} \boxdot \begin{pmatrix} I_{|\sigma|} & 0 \\ 0 & P_\varrho \end{pmatrix}$$

- Easy proof via seaweeds:

A Final Example: Stitching with I

- Consider following task:
  Have permutations $(1, \dots, n) \mapsto (\sigma(1), \dots, \sigma(n))$ and $(1, \dots, n) \mapsto (\varrho(1), \dots, \varrho(n))$
  Want permutation $(1, \dots, n, n+1, \dots, 2n) \mapsto (\sigma(1), \dots, \sigma(n), \varrho(1), \dots, \varrho(n))$
  $\rightsquigarrow$ Prime application: string concatenation
- Claim:

$$P_{\sigma\varrho} = \begin{pmatrix} P_\sigma & 0 \\ 0 & P_\varrho \end{pmatrix} = \begin{pmatrix} P_\sigma & 0 \\ 0 & I_{|\varrho|} \end{pmatrix} \boxdot \begin{pmatrix} I_{|\sigma|} & 0 \\ 0 & P_\varrho \end{pmatrix}$$

- Easy proof via seaweeds:

A Final Example: Stitching with I

- Consider following task:
  Have permutations $(1, \dots, n) \mapsto (\sigma(1), \dots, \sigma(n))$ and $(1, \dots, n) \mapsto (\varrho(1), \dots, \varrho(n))$
  Want permutation $(1, \dots, n, n+1, \dots, 2n) \mapsto (\sigma(1), \dots, \sigma(n), \varrho(1), \dots, \varrho(n))$
  $\rightsquigarrow$ Prime application: string concatenation
- Claim:

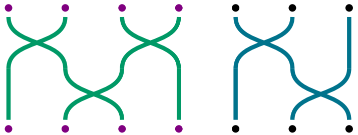
$$P_{\sigma\varrho} = \begin{pmatrix} P_\sigma & 0 \\ 0 & P_\varrho \end{pmatrix} = \begin{pmatrix} P_\sigma & 0 \\ 0 & I_{|\varrho|} \end{pmatrix} \boxdot \begin{pmatrix} I_{|\sigma|} & 0 \\ 0 & P_\varrho \end{pmatrix}$$

- Easy proof via seaweeds:

- Consider following task:
  Have permutations $(1, \dots, n) \mapsto (\sigma(1), \dots, \sigma(n))$ and $(1, \dots, n) \mapsto (\varrho(1), \dots, \varrho(n))$
  Want permutation $(1, \dots, n, n+1, \dots, 2n) \mapsto (\sigma(1), \dots, \sigma(n), \varrho(1), \dots, \varrho(n))$
  $\rightsquigarrow$ Prime application: string concatenation
- Claim:

$$P_{\sigma\varrho} = \begin{pmatrix} P_\sigma & 0 \\ 0 & P_\varrho \end{pmatrix} = \begin{pmatrix} P_\sigma & 0 \\ 0 & I_{|\varrho|} \end{pmatrix} \boxdot \begin{pmatrix} I_{|\sigma|} & 0 \\ 0 & P_\varrho \end{pmatrix}$$

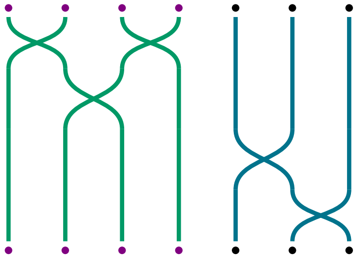
- Easy proof via seaweeds:



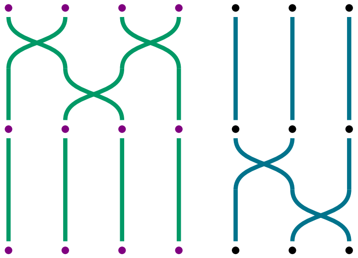
- Consider following task:
  Have permutations $(1,\dots,n) \mapsto (\sigma(1),\dots,\sigma(n))$ and $(1,\dots,n) \mapsto (\varrho(1),\dots,\varrho(n))$
  Want permutation $(1,\dots,n,n+1,\dots,2n) \mapsto (\sigma(1),\dots,\sigma(n),\varrho(1),\dots,\varrho(n))$
  $\rightsquigarrow$ Prime application: string concatenation
- Claim:

$$P_{\sigma\varrho} = \begin{pmatrix} P_\sigma & 0 \\ 0 & P_\varrho \end{pmatrix} = \begin{pmatrix} P_\sigma & 0 \\ 0 & I_{|\varrho|} \end{pmatrix} \boxdot \begin{pmatrix} I_{|\sigma|} & 0 \\ 0 & P_\varrho \end{pmatrix}$$

- Easy proof via seaweeds:



- Consider following task:
  Have permutations $(1, \dots, n) \mapsto (\sigma(1), \dots, \sigma(n))$ and $(1, \dots, n) \mapsto (\varrho(1), \dots, \varrho(n))$
  Want permutation $(1, \dots, n, n+1, \dots, 2n) \mapsto (\sigma(1), \dots, \sigma(n), \varrho(1), \dots, \varrho(n))$
  $\rightsquigarrow$ Prime application: string concatenation
- Claim:
  $$P_{\sigma\varrho} = \begin{pmatrix} P_\sigma & 0 \\ 0 & P_\varrho \end{pmatrix} = \begin{pmatrix} P_\sigma & 0 \\ 0 & I_{|\varrho|} \end{pmatrix} \boxdot \begin{pmatrix} I_{|\sigma|} & 0 \\ 0 & P_\varrho \end{pmatrix}$$
- Easy proof via seaweeds:



- Somewhat weird matrix product (min-plus product of prefix-sum matrices of permutation matrices)
- Seaweed product can be computed efficiently

  $\rightsquigarrow$ Important special case: stitching with I)
- Next, and most importantly: can reinterpret the original DP matrix for ED as prefix-sum matrix of a permutation matrix (after some massaging)

- Somewhat weird matrix product (min-plus product of prefix-sum matrices of permutation matrices)
- Seaweed product can be computed efficiently

  $\rightsquigarrow$ Important special case: stitching with I)
- Next, and most importantly: can reinterpret the original DP matrix for ED as prefix-sum matrix of a permutation matrix (after some massaging)

- Somewhat weird matrix product (min-plus product of prefix-sum matrices of permutation matrices)
- Seaweed product can be computed efficiently

  $\rightsquigarrow$ Important special case: stitching with I)
- Next, and most importantly: can reinterpret the original DP matrix for ED as prefix-sum matrix of a permutation matrix (after some massaging)

Given P, T, need to compute ED between arbitrary substrings
$\rightsquigarrow$ **Goal**: Use seaweeds for speed-ups over standard DP

DP table $e(i, j)$

- $e(i, j) = \mathrm{ED}(P[\,0\,..\,i\,), T[\,0\,..\,j\,))$
- naive: $O(|P| \cdot |T|)$ computation
- kangaroo jumps [LV89]
  $\rightsquigarrow$ time $O(|P| + |T| + k^2)$ for ED
  $\rightsquigarrow$ time $O((|P| + |T|)k)$ for APM

$\rightsquigarrow$

?

- Want to use (min, +)-product (of matrices stored as perm's)

$\rightsquigarrow$ recast ED as shortest path prob?

Given P, T, need to compute ED between arbitrary substrings
$\rightsquigarrow$ **Goal**: Use seaweeds for speed-ups over standard DP

DP table $e(i,j)$

- $e(i,j) = \mathrm{ED}(P[0\,..\,i\,), T[0\,..\,j\,))$
- naive: $O(|P| \cdot |T|)$ computation
- kangaroo jumps [LV89]
  $\rightsquigarrow$ time $O(|P| + |T| + k^2)$ for ED
  $\rightsquigarrow$ time $O((|P| + |T|)k)$ for APM

$\rightsquigarrow$

?

- Want to use (min, +)-product (of matrices stored as perm's)

$\rightsquigarrow$ recast ED as shortest path prob?

Given P, T, need to compute ED between arbitrary substrings
$\rightsquigarrow$ **Goal**: Use seaweeds for speed-ups over standard DP

DP table $e(i, j)$

- $e(i, j) = \mathrm{ED}(P[\,0\,..\,i\,), T[\,0\,..\,j\,))$
- naive: $O(|P| \cdot |T|)$ computation
- kangaroo jumps [LV89]
  $\rightsquigarrow$ time $O(|P| + |T| + k^2)$ for ED
  $\rightsquigarrow$ time $O((|P| + |T|)k)$ for APM

$\rightsquigarrow$

?

- Want to use (min, +)-product (of matrices stored as perm's)

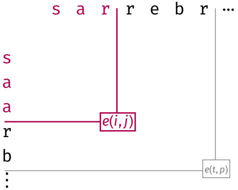
$\rightsquigarrow$ recast ED as shortest path prob?

From DP to Alignment Graphs

Recall classical DP (again):

- Write $e(i,j)$ for the ED of $T[0\,..\,i)$ and $P[0\,..\,j)$
  $\rightsquigarrow$ $e(t,p)$ is the ED of T and P.
- Clearly, $e(i,0) = i$ (delete $T[0\,..\,i)$)
  and $e(0,j) = j$ (insert $P[0\,..\,j)$)
- Observe

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

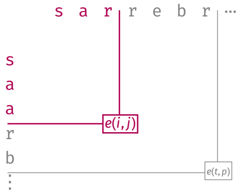


		s	a	r	r	e	b	r	⋯
	0	1	2	3	4	5	6	7	
s	1	0	1	2	3	4	5	6	
a	2	1	0	1	2	3	4	5	
a	3	2	1						
r	4								
b	5								
⋮									

Recall classical DP (again):

- Write $e(i,j)$ for the ED of $T[0..i)$ and $P[0..j)$
  $\rightsquigarrow$ $e(t,p)$ is the ED of T and P.
- Clearly, $e(i,0) = i$ (delete $T[0..i)$)
  and $e(0,j) = j$ (insert $P[0..j)$)
- Observe

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



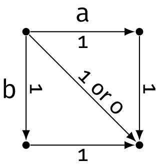
		s	a	r	r	e	b	r	…
	0	1	2	3	4	5	6	7	
s	1	0	1	2	3	4	5	6	
a	2	1	0	1	2	3	4	5	
a	3	2	1						
r	4								
b	5								
⋮									

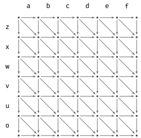
Consider different, but related alignment graph:

- Vertex (i,j) for every $i \in [0..|T|]$, $j \in [0..|P|]$
- Horizontal edge $(i,j) \to (i+1,j)$ of weight 1 (deletion from T)
- Vertical edge $(i,j) \to (i,j+1)$ of weight 1 (insertion in T)
- Diagonal edge $(i,j) \to (i+1,j+1)$ of weight $(T[i] \neq P[j])$ (match/subst)

$\rightsquigarrow$ shortest $(0,0)$ to $(|T|,|P|)$ path is ED of T and P

$\rightsquigarrow$ recover DP by running Dijkstra from $(0,0)$

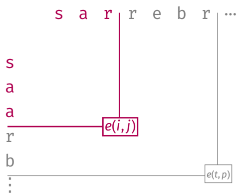




Recall classical DP (again):

- Write $e(i,j)$ for the ED of $T[0..i)$ and $P[0..j)$
  $\rightsquigarrow$ $e(t,p)$ is the ED of T and P.
- Clearly, $e(i,0) = i$ (delete $T[0..i)$)
  and $e(0,j) = j$ (insert $P[0..j)$)
- Observe

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



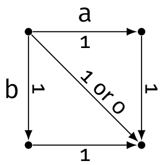
		s	a	r	r	e	b	r	…
	0	1	2	3	4	5	6	7	
s	1	0	1	2	3	4	5	6	
a	2	1	0	1	2	3	4	5	
a	3	2	1						
r	4								
b	5								
⋮									

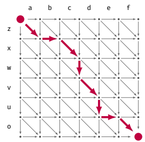
Consider different, but related alignment graph:

- Vertex (i,j) for every $i \in [0..|T|], j \in [0..|P|]$
- Horizontal edge $(i,j) \to (i+1,j)$ of weight 1 (deletion from T)
- Vertical edge $(i,j) \to (i,j+1)$ of weight 1 (insertion in T)
- Diagonal edge $(i,j) \to (i+1,j+1)$ of weight $(T[i] \neq P[j])$ (match/subst)

$\rightsquigarrow$ shortest $(0,0)$ to $(|T|,|P|)$ path is ED of T and P

$\rightsquigarrow$ recover DP by running Dijkstra from $(0,0)$



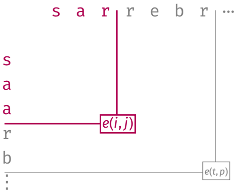


From DP to Alignment Graphs

Recall classical DP (again):

- Write $e(i,j)$ for the ED of $T[0..i)$ and $P[0..j)$
  $\rightsquigarrow$ $e(t,p)$ is the ED of T and P.
- Clearly, $e(i,0) = i$ (delete $T[0..i)$)
  and $e(0,j) = j$ (insert $P[0..j)$)
- Observe

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



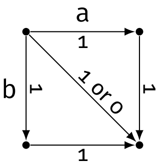
	s	a	r	r	e	b	r	…
0	1	2	3	4	5	6	7	
s 1	0	1	2	3	4	5	6	
a 2	1	0	1	2	3	4	5	
a 3	2	1						
r 4								
b 5								
⋮								

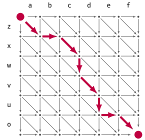
Consider different, but related alignment graph:

- Vertex (i,j) for every $i \in [0..|T|], j \in [0..|P|]$
- Horizontal edge $(i,j) \to (i+1,j)$ of weight 1 (deletion from T)
- Vertical edge $(i,j) \to (i,j+1)$ of weight 1 (insertion in T)
- Diagonal edge $(i,j) \to (i+1,j+1)$ of weight $(T[i] \neq P[j])$ (match/subst)

$\rightsquigarrow$ shortest $(0,0)$ to $(|T|,|P|)$ path is ED of T and P

$\rightsquigarrow$ recover DP by running Dijkstra from $(0,0)$





Alignment Graph of P and T

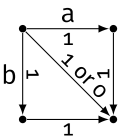
- Vertex (i, j) for every $i \in [0..|T|], j \in [0..|P|]$
- Horizontal edge $(i, j) \to (i+1, j)$ of weight 1 (deletion from T)
- Vertical edge $(i, j) \to (i, j+1)$ of weight 1 (insertion in T)
- Diagonal edge $(i, j) \to (i+1, j+1)$ of weight $(T[i] \neq P[j])$ (match/subst)

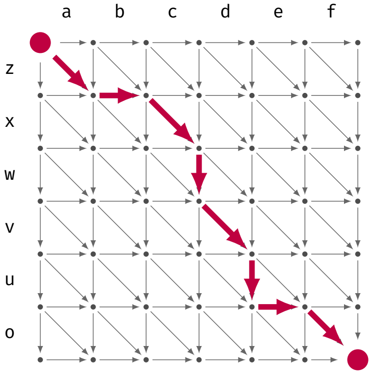
⇝ shortest $(0,0)$ to $(|T|, |P|)$ path is ED of T and P

- shortest (i, j) to (i', j') path is ED of $T[i..i')$ and $P[j..j')$)

⇝ For stitching: Need (just) boundary-to-boundary dist's

- Define $D_{P,T}[a, b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$





$$\mathrm{dist}((0,0),(|T|,|P|)) = \mathrm{ED}(T, P)$$

Alignment Graph of P and T

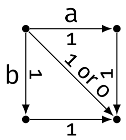
- Vertex (i,j) for every $i \in [0..|T|], j \in [0..|P|]$
- Horizontal edge $(i,j) \to (i+1,j)$ of weight 1 (deletion from T)
- Vertical edge $(i,j) \to (i,j+1)$ of weight 1 (insertion in T)
- Diagonal edge $(i,j) \to (i+1,j+1)$ of weight $(T[i] \neq P[j])$ (match/subst)

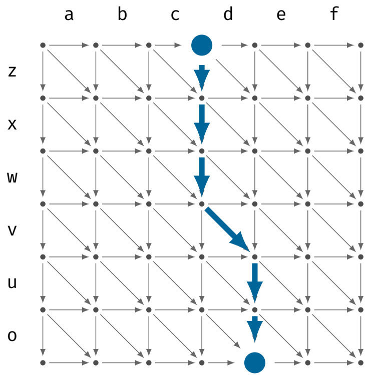
⇝ shortest $(0,0)$ to $(|T|,|P|)$ path is ED of T and P

- shortest (i,j) to (i',j') path is ED of $T[i..i')$ and $P[j..j'))$

⇝ For stitching: Need (just) boundary-to-boundary dist's

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$





$$\mathrm{dist}((i,0),(i',|P|)) = \mathrm{ED}(T[i..i'),P)$$

Alignment Graph of P and T

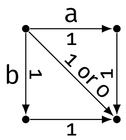
- Vertex (i, j) for every $i \in [0..|T|], j \in [0..|P|]$
- Horizontal edge $(i, j) \to (i+1, j)$ of weight 1 (deletion from T)
- Vertical edge $(i, j) \to (i, j+1)$ of weight 1 (insertion in T)
- Diagonal edge $(i, j) \to (i+1, j+1)$ of weight $(T[i] \neq P[j])$ (match/subst)

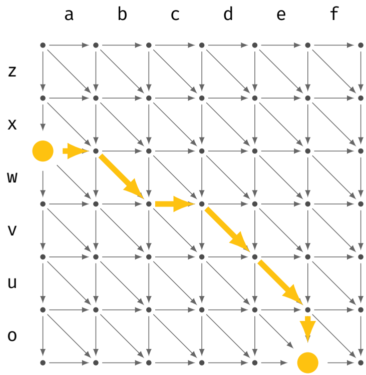
⇝ shortest $(0,0)$ to $(|T|, |P|)$ path is ED of T and P

- shortest (i, j) to (i', j') path is ED of $T[i..i')$ and $P[j..j'))$

⇝ For stitching: Need (just) boundary-to-boundary dist's

- Define $D_{P,T}[a, b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$





$\mathrm{dist}((i, j), (i', j')) = \mathrm{ED}(T[i..i'), P[j..j'))$

Alignment Graph of P and T

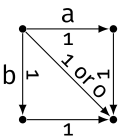
- Vertex (i,j) for every $i \in [0..|T|], j \in [0..|P|]$
- Horizontal edge $(i,j) \to (i+1,j)$ of weight 1 (deletion from T)
- Vertical edge $(i,j) \to (i,j+1)$ of weight 1 (insertion in T)
- Diagonal edge $(i,j) \to (i+1,j+1)$ of weight $(T[i] \neq P[j])$ (match/subst)

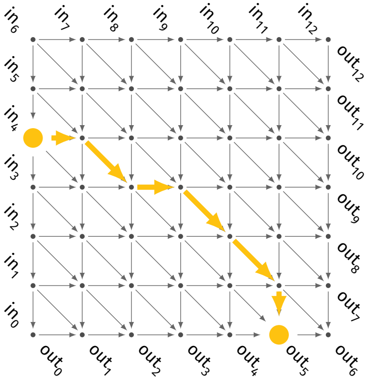
⇝ shortest $(0,0)$ to $(|T|,|P|)$ path is ED of T and P

- shortest (i,j) to (i',j') path is ED of $T[i..i')$ and $P[j..j'))$

⇝ For stitching: Need (just) boundary-to-boundary dist's

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$





$\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \mathrm{ED}(T[\max(a-|P|,0)..\min(b,|T|)),$
$P[\max(|P|-a,0)..\min(|P|+|T|-b,|P|)))$

Alignment Graph of P and T

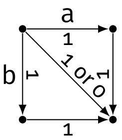
- Vertex (i, j) for every $i \in [\,0\,..\,|T|\,], j \in [\,0\,..\,|P|\,]$
- Horizontal edge $(i, j) \to (i+1, j)$ of weight 1 (deletion from T)
- Vertical edge $(i, j) \to (i, j+1)$ of weight 1 (insertion in T)
- Diagonal edge $(i, j) \to (i+1, j+1)$ of weight $(T[\,i\,] \neq P[\,j\,])$ (match/subst)

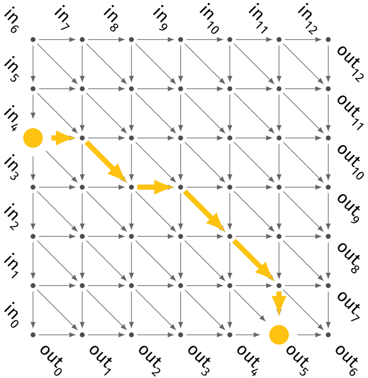
⇝ shortest $(0, 0)$ to $(|T|, |P|)$ path is ED of T and P

- shortest (i, j) to (i', j') path is ED of $T[\,i\,..\,i'\,)$ and $P[\,j\,..\,j')$)

⇝ For stitching: Need (just) boundary-to-boundary dist's

- Define $D_{P,T}[\,a, b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$





$$\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \mathrm{ED}(T[\,\max(a-|P|, 0)\,..\,\min(b, |T|)\,), P[\,\max(|P|-a, 0)\,..\,\min(|P|+|T|-b, |P|)\,))$$

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a, b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$

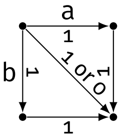
⇝ Hope: $D^{\square}_{P,T}$ is permutation matrix

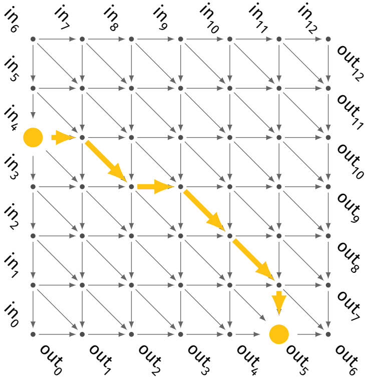
- Problem: Some $D_{P,T}[\,a, b\,]$ are ∞
  ⇝ use undirected edges
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer
  - Border satisfies ($n := |P| + |T|$)

$$D_{P,T} \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[\,a, 0\,] = 0$
- $D_{P,T}[\,a, n\,] = n - a$
- $D_{P,T}[\,n, b\,] = 0$
- $D_{P,T}[\,0, b\,] = b$

Idea: Force boundary values to be correct
⇝ $D'_{P,T}[\,a, b\,] := (D_{P,T}[\,a, b\,] - a + b)/2$





$$\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \mathrm{ED}(T[\max(a - |P|, 0) \,..\, \min(b, |T|)),$$
$$P[\max(|P| - a, 0) \,..\, \min(|P| + |T| - b, |P|)))$$

From DP to Alignment Graphs

Consider Alignment Graph of P and T

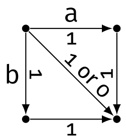
- Define $D_{P,T}[\,a, b\,] := \text{dist}(\text{in}_a, \text{out}_b)$
- ⇝ Hope: $D^{\square}_{P,T}$ is permutation matrix
- Problem: Some $D_{P,T}[\,a, b\,]$ are ∞
  - ⇝ use undirected edges
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer
  - Border satisfies ($n := |P| + |T|$)

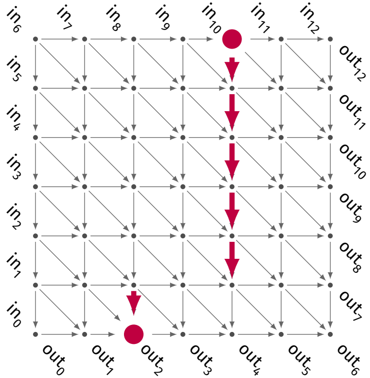
$$D_{P,T} \overset{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

  - $D_{P,T}[\,a, 0\,] = 0$
  - $D_{P,T}[\,a, n\,] = n - a$
  - $D_{P,T}[\,n, b\,] = 0$
  - $D_{P,T}[\,0, b\,] = b$

Idea: Force boundary values to be correct

⇝ $D'_{P,T}[\,a, b\,] := (D_{P,T}[\,a, b\,] - a + b)/2$





$$\text{dist}(\,\text{in}_a, \text{out}_b\,) = \text{ED}(T[\,\max(a - |P|, 0)\,..\,\min(b, |T|)\,),$$
$$P[\,\max(|P| - a, 0)\,..\,\min(|P| + |T| - b, |P|)\,))$$

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a,b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$

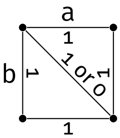
$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

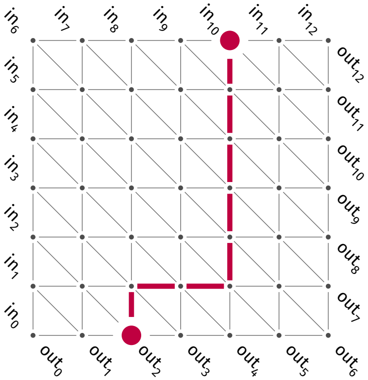
- Problem: Some $D_{P,T}[\,a,b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer
  - Border satisfies ($n := |P| + |T|$)
    - $D_{P,T}[\,a,0\,] = 0$
    - $D_{P,T}[\,a,n\,] = n - a$
    - $D_{P,T}[\,n,b\,] = 0$
    - $D_{P,T}[\,0,b\,] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow D'_{P,T}[\,a,b\,] := (D_{P,T}[\,a,b\,] - a + b)/2$





$$\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) \overset{*}{=} \mathrm{ED}(T[\max(a - |P|, 0) \mathinner{..} \min(b, |T|)),$$
$$P[\max(|P| - a, 0) \mathinner{..} \min(|P| + |T| - b, |P|)))$$

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a,b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$
- Problem: Some $D_{P,T}[\,a,b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

$\rightsquigarrow$ Hope: $D_{P,T}^{\square}$ is permutation matrix

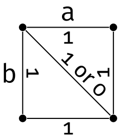
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer
  - Border satisfies ($n := |P| + |T|$)

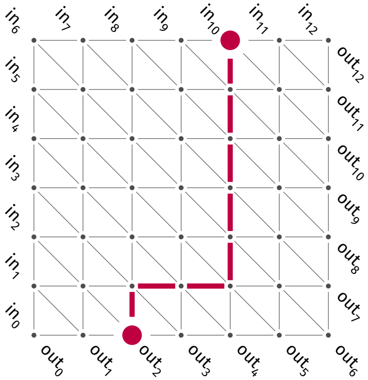
$$D_{P,T} \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \vdots & & & & \vdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

  - $D_{P,T}[\,a,0\,] = 0$
  - $D_{P,T}[\,a,n\,] = n - a$
  - $D_{P,T}[\,n,b\,] = 0$
  - $D_{P,T}[\,0,b\,] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow D'_{P,T}[\,a,b\,] := (D_{P,T}[\,a,b\,] - a + b)/2$





$$\mathrm{dist}(\,\mathrm{in}_a, \mathrm{out}_b\,) \stackrel{*}{=} \mathrm{ED}(T[\,\max(a - |P|, 0) \,..\, \min(b, |T|)\,),$$
$$P[\,\max(|P| - a, 0) \,..\, \min(|P| + |T| - b, |P|)\,))$$

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a, b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$
- Problem: Some $D_{P,T}[\,a, b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

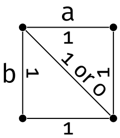
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer
  - Border satisfies ($n := |P| + |T|$)

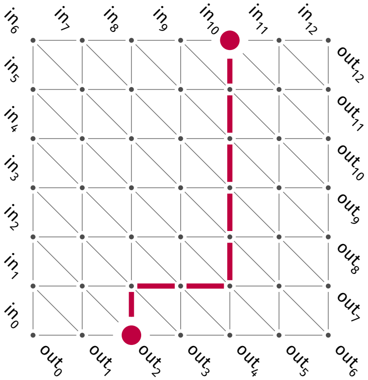
$$D_{P,T} \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[\,a, 0\,] = 0$
- $D_{P,T}[\,a, n\,] = n - a$
- $D_{P,T}[\,n, b\,] = 0$
- $D_{P,T}[\,0, b\,] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow$ $D'_{P,T}[\,a, b\,] := (D_{P,T}[\,a, b\,] - a + b)/2$





$$\mathrm{dist}(\,\mathrm{in}_a, \mathrm{out}_b\,) \stackrel{*}{=} \mathrm{ED}(T[\,\max(a - |P|, 0) \,..\, \min(b, |T|\,)\,),$$
$$P[\,\max(|P| - a, 0) \,..\, \min(|P| + |T| - b, |P|\,)\,)\,)$$

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a, b\,] := \text{dist}(\text{in}_a, \text{out}_b)$
- Problem: Some $D_{P,T}[\,a, b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

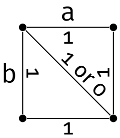
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer ✓
  - Border satisfies ($n := |P| + |T|$)

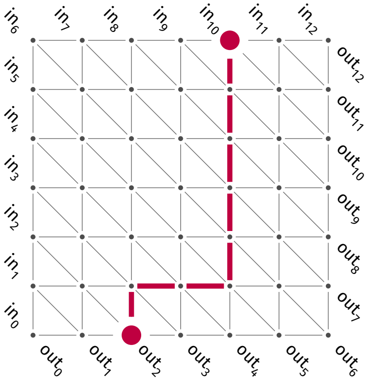
$$D_{P,T} \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

  - $D_{P,T}[\,a, 0\,] = 0$
  - $D_{P,T}[\,a, n\,] = n - a$
  - $D_{P,T}[\,n, b\,] = 0$
  - $D_{P,T}[\,0, b\,] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow$ $D'_{P,T}[\,a, b\,] := (D_{P,T}[\,a, b\,] - a + b)/2$





$$\text{dist}(\,\text{in}_a, \text{out}_b\,) \stackrel{*}{=} \text{ED}(T[\,\max(a - |P|, 0) \,..\, \min(b, |T|\,)\,),$$
$$P[\,\max(|P| - a, 0) \,..\, \min(|P| + |T| - b, |P|\,)\,))$$

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a,b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$
- Problem: Some $D_{P,T}[\,a,b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

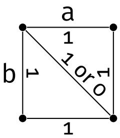
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer ✓
  - Border satisfies ($n := |P| + |T|$)

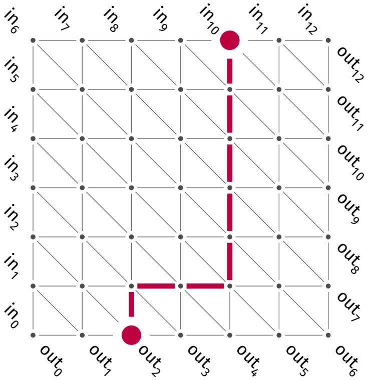
$$D_{P,T} \overset{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[\,a,0\,] = 0$
- $D_{P,T}[\,a,n\,] = n - a$
- $D_{P,T}[\,n,b\,] = 0$
- $D_{P,T}[\,0,b\,] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow D'_{P,T}[\,a,b\,] := (D_{P,T}[\,a,b\,] - a + b)/2$





$$\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) \overset{*}{=} \mathrm{ED}(T[\max(a-|P|,0)\,..\,\min(b,|T|)), P[\max(|P|-a,0)\,..\,\min(|P|+|T|-b,|P|)))$$

From DP to Alignment Graphs

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a, b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$
- Problem: Some $D_{P,T}[\,a, b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

$\rightsquigarrow$ Hope: $D_{P,T}^{\square}$ is permutation matrix

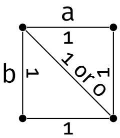
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer ✓
  - Border satisfies ($n := |P| + |T|$)

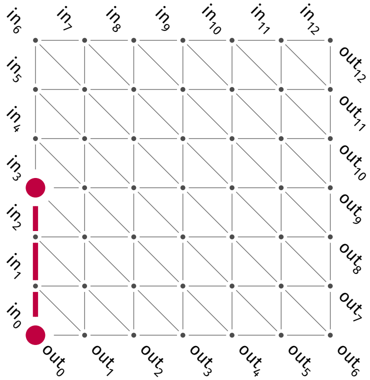
$$D_{P,T} \overset{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[\,a, 0\,] = 0$
- $D_{P,T}[\,a, n\,] = n - a$
- $D_{P,T}[\,n, b\,] = 0$
- $D_{P,T}[\,0, b\,] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow D'_{P,T}[\,a, b\,] := (D_{P,T}[\,a, b\,] - a + b)/2$





$$\mathrm{dist}(\,\mathrm{in}_a, \mathrm{out}_0\,) \overset{*}{=} \mathrm{ED}(T[\,\max(a - |P|, 0)\,..\,0\,),$$
$$P[\,\max(|P| - a, 0)\,..\,|P|\,))$$

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a,b\,] := \mathrm{dist}(\mathrm{in}_a,\mathrm{out}_b)$
- Problem: Some $D_{P,T}[\,a,b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

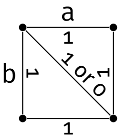
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer ✓
  - Border satisfies ($n := |P| + |T|$)

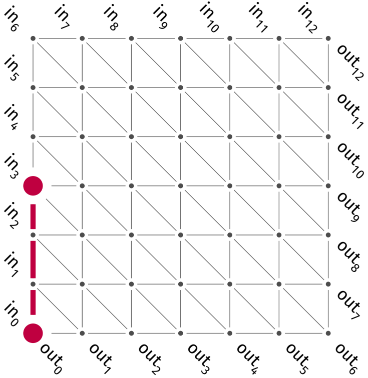
$$D_{P,T} \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \vdots & & & & \vdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[\,a,0\,] = 0$
- $D_{P,T}[\,a,n\,] = n - a$
- $D_{P,T}[\,n,b\,] = 0$
- $D_{P,T}[\,0,b\,] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow D'_{P,T}[\,a,b\,] := (D_{P,T}[\,a,b\,] - a + b)/2$





$\mathrm{dist}(\mathrm{in}_a,\mathrm{out}_0) = \mathrm{ED}(\varepsilon, P[\,\max(|P| - a, 0)\,..\,|P|\,)) = a$

Consider Alignment Graph of P and T

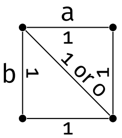
- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$
- Problem: Some $D_{P,T}[a,b]$ are ∞
  $\rightsquigarrow$ use undirected edges

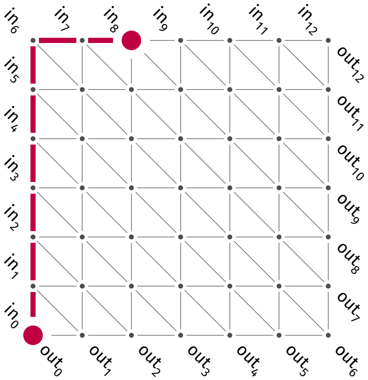
$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer ✓
  - Border satisfies ($n := |P| + |T|$)

$$D_{P,T} \overset{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[a,0] = \cancel{0}\ a$
- $D_{P,T}[a,n] = n - a$
- $D_{P,T}[n,b] = 0$
- $D_{P,T}[0,b] = b$





$\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_0) = a$

Idea: Force boundary values to be correct

$\rightsquigarrow D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a, b\,] := \operatorname{dist}(\mathrm{in}_a, \mathrm{out}_b)$
- Problem: Some $D_{P,T}[\,a, b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

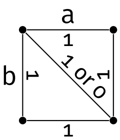
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer ✓
  - Border satisfies ($n := |P| + |T|$)

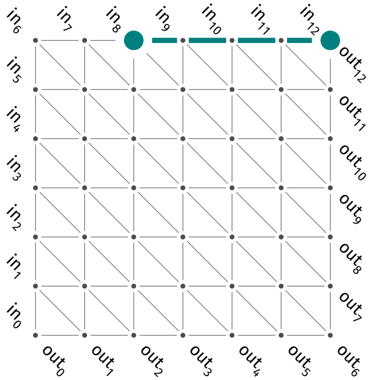
$$D_{P,T} \overset{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[\,a, 0\,] = \cancel{0}\ a$
- $D_{P,T}[\,a, n\,] = n - a$
- $D_{P,T}[\,n, b\,] = 0$
- $D_{P,T}[\,0, b\,] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow D'_{P,T}[\,a, b\,] := (D_{P,T}[\,a, b\,] - a + b)/2$





$\operatorname{dist}(\,\mathrm{in}_a, \mathrm{out}_n\,) = n - a$

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a, b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$
- Problem: Some $D_{P,T}[\,a, b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

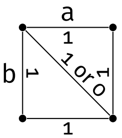
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer ✓
  - Border satisfies ($n := |P| + |T|$)

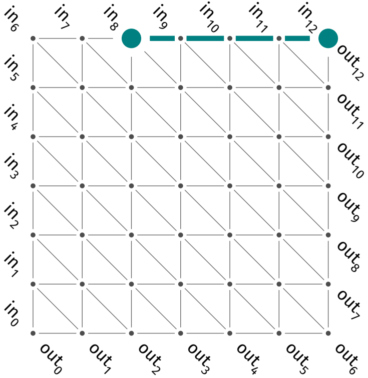
$$D_{P,T} \overset{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[\,a, 0\,] = \cancel{0}\ a$
- $D_{P,T}[\,a, n\,] = n - a$
- $D_{P,T}[\,n, b\,] = 0$
- $D_{P,T}[\,0, b\,] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow D'_{P,T}[\,a, b\,] := (D_{P,T}[\,a, b\,] - a + b)/2$





$\mathrm{dist}(\,\mathrm{in}_a, \mathrm{out}_n\,) = n - a$

Consider Alignment Graph of P and T

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$
- Problem: Some $D_{P,T}[a,b]$ are ∞
  $\rightsquigarrow$ use undirected edges

$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

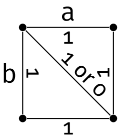
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer ✓
  - Border satisfies ($n := |P| + |T|$)

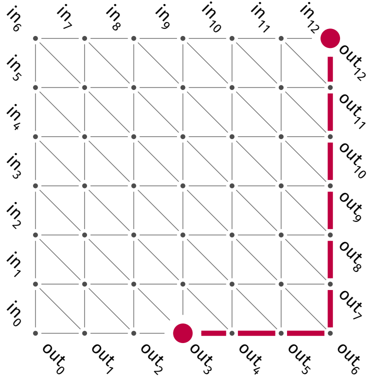
$$D_{P,T} \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[a,0] = θ\ a$
- $D_{P,T}[a,n] = n - a$
- $D_{P,T}[n,b] = θ\ n - b$
- $D_{P,T}[0,b] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$





$\mathrm{dist}(\mathrm{in}_n, \mathrm{out}_b) = n - b$

Consider Alignment Graph of P and T

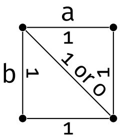
- Define $D_{P,T}[\,a, b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$
- Problem: Some $D_{P,T}[\,a, b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

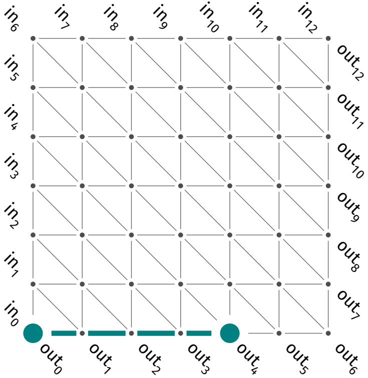
$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer ✓
  - Border satisfies ($n := |P| + |T|$)

$$D_{P,T} \overset{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[\,a, 0\,] = \cancel{0}\ a$
- $D_{P,T}[\,a, n\,] = n - a$
- $D_{P,T}[\,n, b\,] = \cancel{0}\ n{-}b$
- $D_{P,T}[\,0, b\,] = b$





$\mathrm{dist}(\,\mathrm{in}_0, \mathrm{out}_b\,) = b$

Idea: Force boundary values to be correct

$\rightsquigarrow D'_{P,T}[\,a, b\,] := (D_{P,T}[\,a, b\,] - a + b)/2$

Consider Alignment Graph of P and T

- Define $D_{P,T}[\,a, b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$
- Problem: Some $D_{P,T}[\,a, b\,]$ are ∞
  $\rightsquigarrow$ use undirected edges

$\rightsquigarrow$ Hope: $D^{\square}_{P,T}$ is permutation matrix

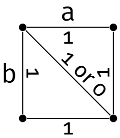
- Suffices to show (recall earlier slide):
  - $D_{P,T}$ is non-negative and integer ✓
  - Border satisfies ($n := |P| + |T|$)

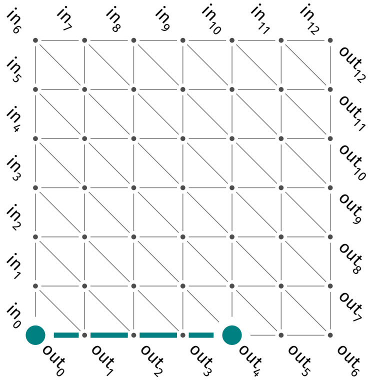
$$D_{P,T} \overset{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

- $D_{P,T}[\,a, 0\,] = \cancel{0}\ a$
- $D_{P,T}[\,a, n\,] = n - a$
- $D_{P,T}[\,n, b\,] = \cancel{0}\ n{-}b$
- $D_{P,T}[\,0, b\,] = b$

Idea: Force boundary values to be correct

$\rightsquigarrow$ $D'_{P,T}[\,a, b\,] := (D_{P,T}[\,a, b\,] - a + b)/2$





$$\mathrm{dist}(\,\mathrm{in}_a, \mathrm{out}_b\,) \overset{*}{=} \mathrm{ED}(T[\,\max(a - |P|, 0)\,..\,\min(b, |T|\,)\,),$$
$$P[\,\max(|P| - a, 0)\,..\,\min(|P| + |T| - b, |P|\,)\,))$$

Consider Alignment Graph of P and T

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix

- Suffices to show (recall earlier slide):
  - $D'_{P,T}$ is non-negative and **integer**
  - Border satisfies ($n := |P| + |T|$)

$$D'_{P,T} \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

Have:

- $D'_{P,T}[a,0] = (a - a + 0)/2 = 0$
- $D'_{P,T}[n,b] = ((n-b) - n + b)/2 = 0$
- $D'_{P,T}[a,n] = ((n-a) - a + n)/2 = n - a$
- $D'_{P,T}[0,b] = (b - 0 + b)/2 = b$

Consider Alignment Graph of P and T

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix

- Suffices to show (recall earlier slide):
  - $D'_{P,T}$ is non-negative and **integer**
  - Border satisfies ($n := |P| + |T|$)

$$D'_{P,T} \overset{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix}$$

Have:

- $D'_{P,T}[a,0] = (a - a + 0)/2 = 0$
- $D'_{P,T}[n,b] = ((n-b) - n + b)/2 = 0$
- $D'_{P,T}[a,n] = ((n-a) - a + n)/2 = n - a$
- $D'_{P,T}[0,b] = (b - 0 + b)/2 = b$

Consider Alignment Graph of P and T

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix

- Suffices to show (recall earlier slide):
  - $D'_{P,T}$ is non-negative and **integer**
  - Border satisfies ($n := |P| + |T|$)

$$D'_{P,T} \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix} \checkmark$$

Have:

- $D'_{P,T}[a,0] = (a - a + 0)/2 = 0$
- $D'_{P,T}[n,b] = ((n-b) - n + b)/2 = 0$
- $D'_{P,T}[a,n] = ((n-a) - a + n)/2 = n - a$
- $D'_{P,T}[0,b] = (b - 0 + b)/2 = b$

Consider Alignment Graph of P and T

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix

- Suffices to show (recall earlier slide):
  - $D'_{P,T}$ is non-negative and **integer**
  - Border satisfies ($n := |P| + |T|$)

$$D'_{P,T} \stackrel{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix} \checkmark$$

Have:

- $D'_{P,T}[a,0] = (a - a + 0)/2 = 0$
- $D'_{P,T}[n,b] = ((n-b) - n + b)/2 = 0$
- $D'_{P,T}[a,n] = ((n-a) - a + n)/2 = n - a$
- $D'_{P,T}[0,b] = (b - 0 + b)/2 = b$

Consider Alignment Graph of P and T

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix

- Suffices to show (recall earlier slide):
  - $D'_{P,T}$ is non-negative and **integer** $\rightsquigarrow$ Problem!
  - Border satisfies ($n := |P| + |T|$)

$$D'_{P,T} \overset{!}{=} \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & & & & 0 \\ \cdots & & & & \cdots \\ n-1 & & & & 0 \\ n & n-1 & \cdots & 1 & 0 \end{pmatrix} \checkmark$$

Have:

- $D'_{P,T}[a,0] = (a - a + 0)/2 = 0$
- $D'_{P,T}[n,b] = ((n-b) - n + b)/2 = 0$
- $D'_{P,T}[a,n] = ((n-a) - a + n)/2 = n - a$
- $D'_{P,T}[0,b] = (b - 0 + b)/2 = b$

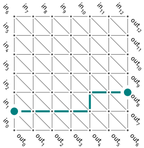
- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix $\rightsquigarrow$ Problem: $D'^{\square}_{P,T}[a,b]$ not integer

$$D_{P,T}[a,b] = \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \mathrm{dist}((x_a, y_a), (x_b, y_b))$$

$$= \begin{cases} |b-a| = |x_a - x_b| + |y_a - y_b| & \text{if } b \le a - |P| \text{ or } b \ge a + |T| \text{ (out is left or above of in, no diag edges)} \\ \max(|x_a - x_b|, |y_a - y_b|) & \text{otherwise (take as many diag's as possible; save if char's equal;} \\ \quad - \mathrm{LCS}(P_{a,b}, T_{a,b}) & \text{max savings: longest common subseq. (lcs) of corresp. parts of } P/T) \end{cases}$$

- Every character subst. flips parity of $\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ $\rightsquigarrow$ "forbid subst's" / increase cost to 2



$\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = |b-a|$

good ($|b-a| - a + b$ always divisible by 2)

$\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \max(|x_a - x_b|, |y_a - y_b|) - \mathrm{LCS}(P_{a,b}, T_{a,b})$

red: cost 1 (bad), green: cost 0 (saving of 2, good)

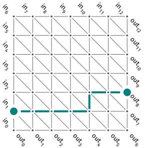
- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix $\rightsquigarrow$ Problem: $D'^{\square}_{P,T}[a,b]$ not integer

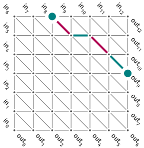
$$D_{P,T}[a,b] = \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \mathrm{dist}((x_a, y_a), (x_b, y_b))$$

$$= \begin{cases} |b-a| = |x_a - x_b| + |y_a - y_b| & \text{if } b \le a - |P| \text{ or } b \ge a + |T| \text{ (out is left or above of in, no diag edges)} \\ \max(|x_a - x_b|, |y_a - y_b|) & \text{otherwise (take as many diag's as possible; save if char's equal;} \\ \quad - \mathrm{LCS}(P_{a,b}, T_{a,b}) & \text{max savings: longest common subseq. (lcs) of corresp. parts of } P/T\text{)} \end{cases}$$

- Every character subst. flips parity of $\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ $\rightsquigarrow$ "forbid subst's" / increase cost to 2



dist ($\mathrm{in}_a, \mathrm{out}_b$) = $|b - a|$
good ($|b-a| - a + b$ always divisible by 2)



dist ($\mathrm{in}_a, \mathrm{out}_b$) = $\max(|x_a - x_b|, |y_a - y_b|) - \mathrm{LCS}(P_{a,b}, T_{a,b})$
red: cost 1 (bad), green: cost 0 (saving of 2, good)

Substitutions Are Too Cheap!

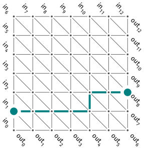
- Define $D_{P,T}[\,a,b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[\,a,b\,] := (D_{P,T}[\,a,b\,] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix $\rightsquigarrow$ Problem: $D'^{\square}_{P,T}[\,a,b\,]$ not integer

$$D_{P,T}[\,a,b\,] = \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \mathrm{dist}((x_a, y_a), (x_b, y_b))$$

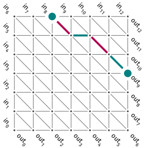
$$= \begin{cases} |b-a| = |x_a - x_b| + |y_a - y_b| & \text{if } b \le a - |P| \text{ or } b \ge a + |T| \text{ (out is left or above of in, no diag edges)} \\ \max(|x_a - x_b|, |y_a - y_b|) & \text{otherwise (take as many diag's as possible; save if char's equal;} \\ \quad - \mathrm{LCS}(P_{a,b}, T_{a,b}) & \text{max savings: longest common subseq. (lcs) of corresp. parts of } P/T) \end{cases}$$

- Every character subst. flips parity of $\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ $\rightsquigarrow$ “forbid subst’s” / increase cost to 2



$\mathrm{dist}(\,\mathrm{in}_a, \mathrm{out}_b\,) = |b - a|$

good ($|b-a| - a + b$ always divisible by 2)



$\mathrm{dist}(\,\mathrm{in}_a, \mathrm{out}_b\,) = \max(|x_a - x_b|, |y_a - y_b|) - \mathrm{LCS}(P_{a,b}, T_{a,b})$

red: cost 1 (bad), green: cost 0 (saving of 2, good)

The Deletion Distance to the Rescue!

- Define $D_{P,T}[\,a,b\,] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[\,a,b\,] := (D_{P,T}[\,a,b\,] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix $\rightsquigarrow$ Problem: $D'^{\square}_{P,T}[\,a,b\,]$ not integer

- Every character subst. flips parity of $\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ $\rightsquigarrow$ “forbid subst’s” / increase cost to 2

Deletion Distance (DD)

Min number of insertions or deletions of characters to transform T into P.

Embedding DD into ED

Write $S^{\$} := S[\,0\,]\$\,S[\,1\,]\$ \cdots S[\,|S|\,]\$$. Then $\mathrm{DD}(P^{\$}, T^{\$}) = 2\mathrm{ED}(P,T)$.

- In alignment graph: corresponds to removing all diag edges of weight 1
  (and adding vertices/edges corresponding to \$)

$$D_{P,T}[\,a,b\,] = \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \mathrm{dist}((x_a,y_a),(x_b,y_b))$$

$$= \begin{cases} |b-a| = |x_a - x_b| + |y_a - y_b| & \text{if } b \le a - |P| \text{ or } b \ge a + |T| \text{ (out is left or above of in, no diag edges)} \end{cases}$$

The Deletion Distance to the Rescue!

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix $\rightsquigarrow$ Problem: $D'^{\square}_{P,T}[a,b]$ not integer

- Every character subst. flips parity of $\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ $\rightsquigarrow$ “forbid subst’s” / increase cost to 2

Deletion Distance (DD)

Min number of insertions or deletions of characters to transform T into P.

Embedding DD into ED

Write $S^{\$} := S[0]\$S[1]\$ \cdots S[|S|]\$$. Then $\mathrm{DD}(P^{\$}, T^{\$}) = 2\mathrm{ED}(P,T)$.

- In alignment graph: corresponds to removing all diag edges of weight 1
  (and adding vertices/edges corresponding to \$)

$D_{P,T}[a,b] = \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \mathrm{dist}((x_a, y_a), (x_b, y_b))$

$= \begin{cases} |b-a| = |x_a - x_b| + |y_a - y_b| & \text{if } b \le a - |P| \text{ or } b \ge a + |T| \text{ (out is left or above of in, no diag edges)} \end{cases}$

The Deletion Distance to the Rescue!

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix $\rightsquigarrow$ Problem: $D'^{\square}_{P,T}[a,b]$ not integer

- Every character subst. flips parity of $\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ $\rightsquigarrow$ "forbid subst's" / increase cost to 2

Deletion Distance (DD)

Min number of insertions or deletions of characters to transform T into P.

Embedding DD into ED

Write $S^{\$} := S[0]\$S[1]\$ \cdots S[|S|]\$$. Then $\mathrm{DD}(P^{\$}, T^{\$}) = 2\mathrm{ED}(P,T)$.

- In alignment graph: corresponds to removing all diag edges of weight 1
  (and adding vertices/edges corresponding to \$)

$$D_{P,T}[a,b] = \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \mathrm{dist}((x_a, y_a), (x_b, y_b))$$

$$= \begin{cases} |b-a| = |x_a - x_b| + |y_a - y_b| & \text{if } b \le a - |P| \text{ or } b \ge a + |T| \text{ (out is left or above of in, no diag edges)} \\ |x_a - x_b| + |y_a - y_b| & \text{otherwise (diag edge only when char's equal; saves 2 per diag} \\ \quad - 2\mathrm{LCS}(P_{a,b}, T_{a,b}) & \text{max savings: longest common subseq. (lcs) of corresp. parts of } P/T\text{)} \end{cases}$$

- Define $D_{P,T}[a,b] := \text{dist}(\text{in}_a, \text{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix $\rightsquigarrow$ Problem: $D'^{\square}_{P,T}[a,b]$ not integer

- Every character subst. flips parity of $\text{dist}(\text{in}_a, \text{out}_b)$ $\rightsquigarrow$ "forbid subst's" / increase cost to 2

Deletion Distance (DD)

Min number of insertions or deletions of characters to transform T into P.

Embedding DD into ED

Write $S^{\$} := S[0]\$S[1]\$ \cdots S[|S|]\$$. Then $\text{DD}(P^{\$}, T^{\$}) = 2\text{ED}(P,T)$.

- In alignment graph: corresponds to removing all diag edges of weight 1
  (and adding vertices/edges corresponding to \$)

$$D_{P,T}[a,b] = \text{dist}(\text{in}_a, \text{out}_b) = \text{dist}((x_a, y_a), (x_b, y_b))$$

$$= \begin{cases} |b-a| = |x_a - x_b| + |y_a - y_b| & \text{if } b \le a - |P| \text{ or } b \ge a + |T| \text{ (out is left or above of in, no diag edges)} \\ |x_a - x_b| + |y_a - y_b| & \text{otherwise (diag edge only when char's equal; saves 2 per diag} \\ \quad - 2\text{LCS}(P_{a,b}, T_{a,b}) & \text{max savings: longest common subseq. (lcs) of corresp. parts of } P/T\text{)} \end{cases}$$

The Deletion Distance to the Rescue!

- Define $D_{P,T}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P,T}[a,b] := (D_{P,T}[a,b] - a + b)/2$

$\rightsquigarrow$ Hope: $D'^{\square}_{P,T}$ is permutation matrix $\rightsquigarrow$ Problem: $D'^{\square}_{P,T}[a,b]$ not integer

- Every character subst. flips parity of $\mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ $\rightsquigarrow$ "forbid subst's" / increase cost to 2

Deletion Distance (DD)

Min number of insertions or deletions of characters to transform T into P.

Embedding DD into ED

Write $S^{\$} := S[0]\$S[1]\$ \cdots S[|S|]\$$. Then $\mathrm{DD}(P^{\$}, T^{\$}) = 2\mathrm{ED}(P,T)$.

- In alignment graph: corresponds to removing all diag edges of weight 1
(and adding vertices/edges corresponding to \$)

$$D_{P,T}[a,b] = \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b) = \mathrm{dist}((x_a, y_a), (x_b, y_b))$$

$$= \begin{cases} |b-a| = |x_a - x_b| + |y_a - y_b| & \text{if } b \le a - |P| \text{ or } b \ge a + |T| \text{ (out is left or above of in, no diag edges)} \\ |x_a - x_b| + |y_a - y_b| & \text{otherwise (diag edge only when char's equal; saves 2 per diag} \\ \quad - 2\mathrm{LCS}(P_{a,b}, T_{a,b}) & \text{ max savings: longest common subseq. (lcs) of corresp. parts of } P/T) \end{cases}$$

Main Result

Consider (deletion distance) alignment graph of $P^\$$ and $T^\$$;
define $D_{P\$,T\$}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P\$,T\$}[a,b] := (D_{P\$,T\$}[a,b] - a + b)/2$.
Then, $D'^{\square}_{P\$,T\$}$ is a permutation matrix.

- Can (easily) show: (min, +)-product of $D'_{P\$,T\$}$ matrices ("stitching") yields deletion distances for concatenated strings
  $\rightsquigarrow$ Can use seaweed product for fast computation
- Some technicalities (skipped):
  - stitching requires some overlap in at least one string
  - want to compute only $\approx k$ diagonals; requires a "restriction" operation
  - formally defining the object we use to represent pairs of strings in DPM
  - ...

Main Result

Consider (deletion distance) alignment graph of $P^{\$}$ and $T^{\$}$;
define $D_{P\$,T\$}[a,b] := \mathrm{dist}(\mathrm{in}_a, \mathrm{out}_b)$ and $D'_{P\$,T\$}[a,b] := (D_{P\$,T\$}[a,b] - a + b)/2$.
Then, $D'^{\square}_{P\$,T\$}$ is a permutation matrix.

- Can (easily) show: (min, +)-product of $D'_{P\$,T\$}$ matrices (“stitching”) yields deletion distances for concatenated strings
  $\rightsquigarrow$ Can use seaweed product for fast computation
- Some technicalities (skipped):
  - stitching requires some overlap in at least one string
  - want to compute only $\approx k$ diagonals; requires a “restriction” operation
  - formally defining the object we use to represent pairs of strings in DPM
  - ...

[CKoW'22]

Dynamic Puzzle Matching

Input: integer k, family F of strings w/ special string Q, s.t. $\sum_{F\in \mathrm{F}} \mathrm{ED}(F, Q) = O(k)$.

Maintain: sequence $I = (U_1, V_1)\cdots(U_z, V_z)$ of pairs from F^2

Updates: Insertions and Deletions of pairs in I

Queries: Compute k-edit occ's of $U_1 \cdots U_z$ in $V_1 \cdots V_z$

Main Result: After $\tilde{O}(k^3)$ preprocessing, updates and queries in time $\tilde{O}(k)$

General Idea

1. Store edit distance information for each pair (U_i, V_i)
   $\rightsquigarrow$ Seaweed/permutation matrix of (U_i, V_i) allow this in $O(k)$ space
2. In preprocessing, build matrix for every possible pair of strings from family F
3. Use seaweed product of [Tis'07,'15] to compose the information of different pairs
   $\tilde{O}(k)$ per stitch
4. Use BST on top to support update queries

What about the weighted setting?

What about the weighted setting?

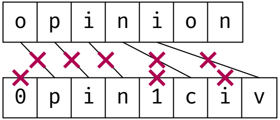
> **10. The weighted case.** In the weighted case, deletions of different characters and the various substitutions may have differing costs, but, by way of normalization, all will be required to have cost at least 1.
>
> The approximate matches with cost $\leq k$ can be found using essentially the same algorithm; the only change needed is to the Landau–Vishkin algorithm to take into account the differing costs. The details are left to the reader.

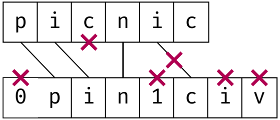
—Cole, Hariharan, 2002

ED(*X*, *Y*)

Edit Distance

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





ED(0PIN1CIV, OPINION) = 5

ED(0PIN1CIV, PICNIC) = 5

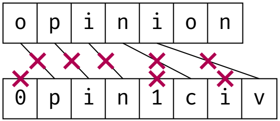
$ED^w(X, Y)$

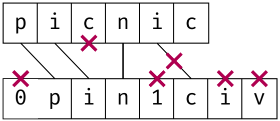
Weighted Edit Distance

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 $w(x, y)$.

$w(\mathtt{0}, \mathrm{O}) := 1$ $w(\mathtt{1}, \mathrm{I}) := 1$ $w(\mathrm{C}, \mathrm{O}) := 1$ $w(*, *) := 2$ $w(*, \varepsilon) := 1$ $w(\varepsilon, *) := 10$





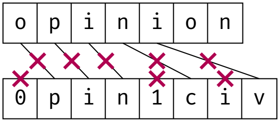
$\mathrm{ED}^w(X, Y)$

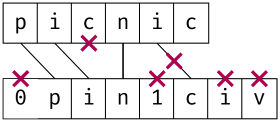
Weighted Edit Distance

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 $w(x, y)$.

$w(0, \mathrm{O}) := 1 \quad w(1, \mathrm{I}) := 1 \quad w(\mathrm{C}, \mathrm{O}) := 1 \qquad w(*, *) := 2 \quad w(*, \varepsilon) := 1 \quad w(\varepsilon, *) := 10$





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

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

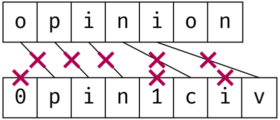
$ED^w(X, Y)$

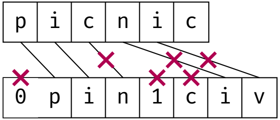
Weighted Edit Distance

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 $w(x, y)$.

$w(\mathtt{0}, \mathtt{O}) := 1 \quad w(\mathtt{1}, \mathtt{I}) := 1 \quad w(\mathtt{C}, \mathtt{O}) := 1 \qquad w(*, *) := 2 \quad w(*, \varepsilon) := 1 \quad w(\varepsilon, *) := 10$





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

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

How fast can we compute the (weighted) edit distance of two strings?

$\rightsquigarrow O(n^2)$ (recalled this earlier!)

How fast can we compute the (weighted) edit distance of two strings?

$\rightsquigarrow O(n^2)$ (recalled this earlier!)

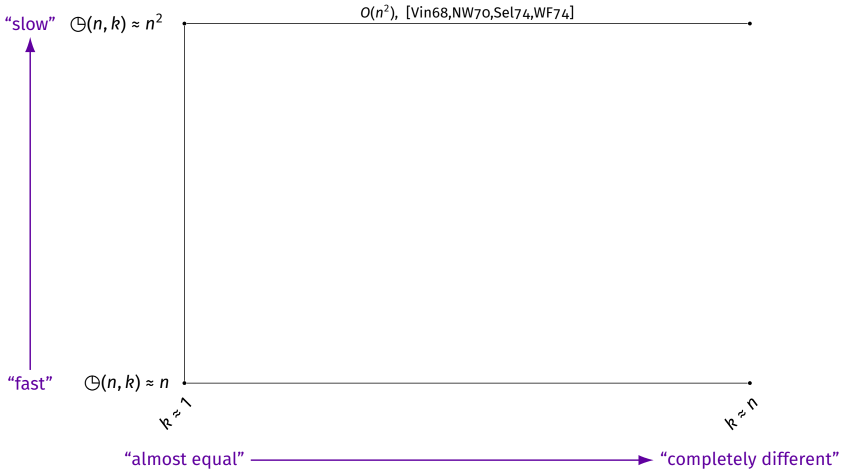
For Pattern Matching, enough to compute (weighted) edit distance if small (at most k)

How fast can we compute the (weighted) edit distance of two strings?

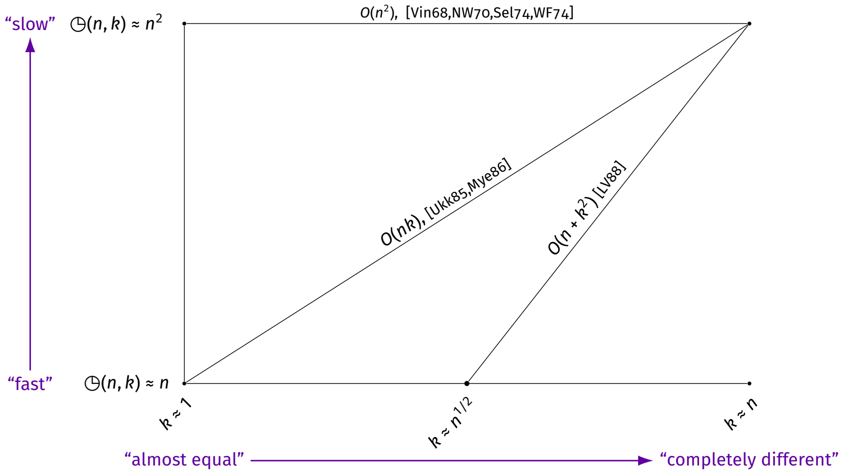
$\rightsquigarrow O(n^2)$ (recalled this earlier!)

For Pattern Matching, enough to compute (weighted) edit distance if small (at most k)

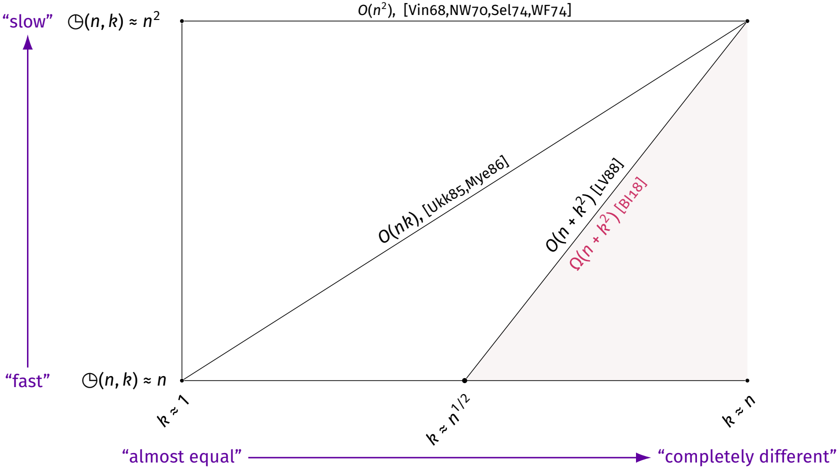
$\rightsquigarrow$ Bounded (Weighted) Edit Distance



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



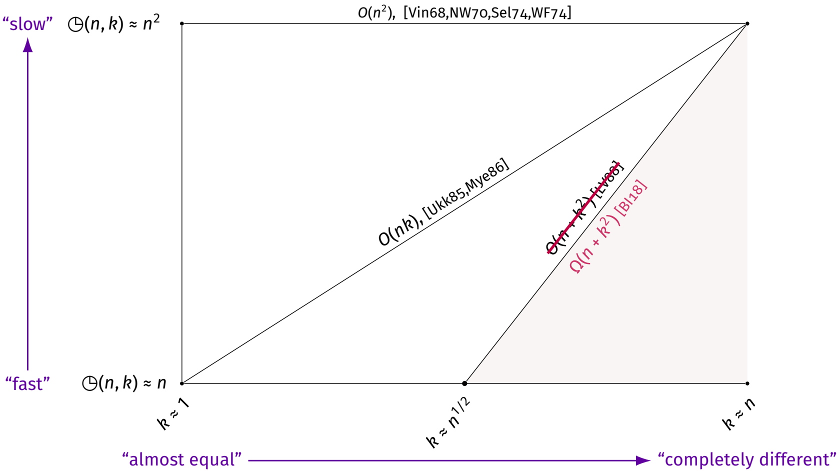
Existing algorithms for Edit Distance $\mathrm{ED}(X, Y)$, where $|X|, |Y| \le n$



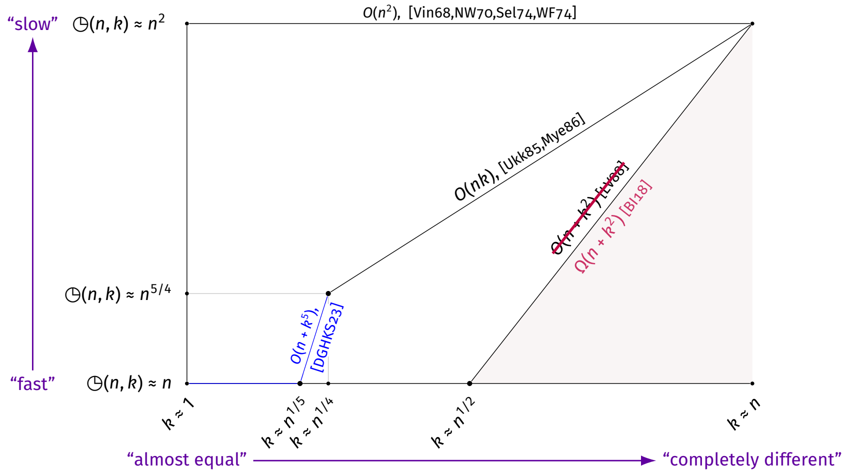
Existing algorithms for Edit Distance $\mathrm{ED}(X, Y)$, where $|X|, |Y| \le n$

What about Weighted Edit Distance?

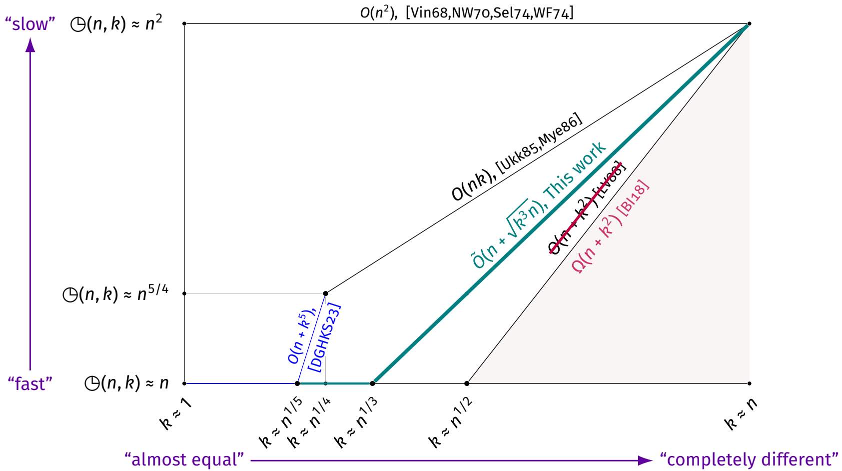
Algorithms for (Bounded) Weighted Edit Distance



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

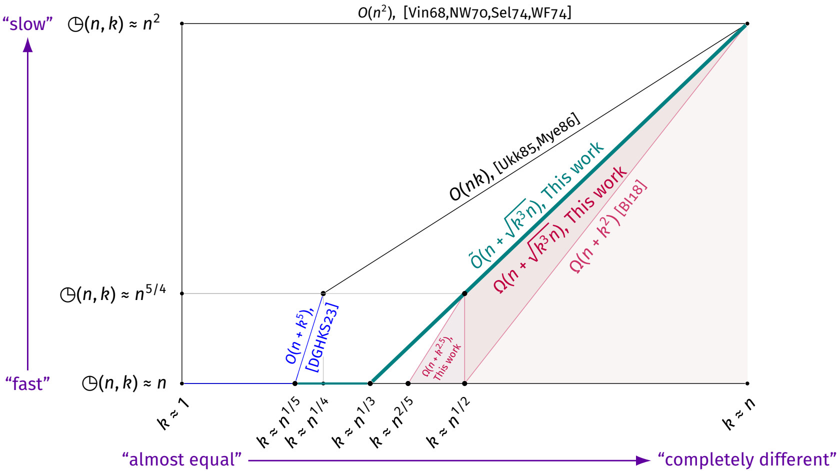


Existing algorithms for Weighted Edit Distance $\mathrm{ED}^w(X, Y)$, where $|X|, |Y| \le n$



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

Algorithms for (Bounded) Weighted Edit Distance



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

[CaKoW'23]

Main Theorem 1 (Upper Bound)

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 $O(n + \sqrt{nk^3}\log^3 n)$

[CaKoW'23]

Main Theorem 2 (Lower Bound)

Assuming the APSP Hypothesis and for $\sqrt{n} \le k \le n$,

Main Theorem 1 is tight (up to $n^{o(1)}$-factors)

$\mathrm{ED}^w_{\leq k}(X, Y)$

Bounded Weighted Edit Distance

Min cost of transforming X to Y using character edits (if it is at most k), where:

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

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

Bounded Weighted Edit Distance

Min cost of transforming X to Y using character edits (if it is at most k), where:

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

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

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

Bounded Weighted Edit Distance

Min cost of transforming X to Y using character edits (if it is at most k), where:

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

- Justified Assumption: w is normalized, $w(x, y) \ge 1$ for all $x \ne y$.

  $\rightsquigarrow$ Have (via $O(n + k^2)$ algo): Alignment $A : X \rightsquigarrow\!\!\!\!\twoheadrightarrow Y$ of unweighted cost $\le k$

$\mathrm{ED}^w_{\le k}(X, Y)$

Bounded Weighted Edit Distance

Min cost of transforming X to Y using character edits (if it is at most k), where:

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

- Justified Assumption: w is normalized, $w(x, y) \ge 1$ for all $x \ne y$.
  - $\rightsquigarrow$ Have (via $O(n + k^2)$ algo): Alignment $A : X \twoheadrightarrow Y$ of unweighted cost $\le k$
  - $\rightsquigarrow$ A aligns edit-free most of X with Y [DGHKS23]

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

Bounded Weighted Edit Distance

Min cost of transforming X to Y using character edits (if it is at most k), where:

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

- Justified Assumption: w is normalized, $w(x, y) \ge 1$ for all $x \ne y$.
  - ⇝ Have (via $O(n + k^2)$ algo): Alignment $A : X \rightsquigarrow\!\!\!\twoheadrightarrow Y$ of unweighted cost $\le k$
  - ⇝ A aligns edit-free most of X with Y [DGHKS23]

[DGHKS23]

Theorem (Universal Kernel)

Can trim X and Y to length-$O(k^4)$ strings X', Y' with $\mathrm{ED}^{w}_{\le k}(X, Y) = \mathrm{ED}^{w}_{\le k}(X', Y')$.

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

Bounded Weighted Edit Distance

Min cost of transforming X to Y using character edits (if it is at most k), where:

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

- Justified Assumption: w is normalized, $w(x, y) \ge 1$ for all $x \ne y$.

$\rightsquigarrow$ Have (via $O(n + k^2)$ algo): Alignment $A : X \rightsquigarrow\!\!\!\twoheadrightarrow Y$ of unweighted cost $\le k$

$\rightsquigarrow$ A aligns edit-free most of X with Y [DGHKS23]

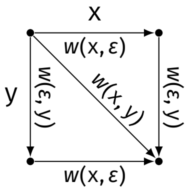
[DGHKS23]

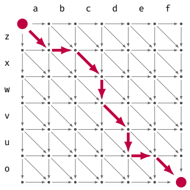
Theorem (Universal Kernel)

Can trim X and Y to length-$O(k^4)$ strings X', Y' with $\mathrm{ED}^{w}_{\le k}(X, Y) = \mathrm{ED}^{w}_{\le k}(X', Y')$.
X' and Y' consist in $O(k^2)$ pieces of length $O(k^2)$ and with period $O(k)$ each

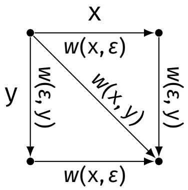
- X and Y consist in $O(k^2)$ pieces of length $O(k^2)$ and with period $O(k)$ each
- Idea: Use (directed) alignment graph AG of X and Y (seaweeds don't work here)
- Idea^2: Trim AG to $O(k)$ diagonals $\rightsquigarrow$ $\mathrm{ED}^w_{\le k}(X, Y)$ is distance $(0, 0) \rightsquigarrow (|X|, |Y|)$
- Idea^3: Split AG according to structure of X and Y
  Compute all b-to-b dist [Klein05] + stitch together results ((min, +)-product [SMAWK87])

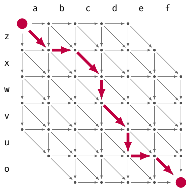
- X and Y consist in $O(k^2)$ pieces of length $O(k^2)$ and with period $O(k)$ each
- Idea: Use (directed) alignment graph AG of X and Y (seaweeds don't work here)
  $\rightsquigarrow$ $\mathrm{ED}^w(X,Y)$ is distance $(0,0) \rightsquigarrow (|X|,|Y|)$
- Idea^2: Trim AG to $O(k)$ diagonals $\rightsquigarrow$ $\mathrm{ED}^w_{\leq k}(X,Y)$ is distance $(0,0) \rightsquigarrow (|X|,|Y|)$
- Idea^3: Split AG according to structure of X and Y
  Compute all b-to-b dist [Klein05] + stitch together results ((min, +)-product [SMAWK87])



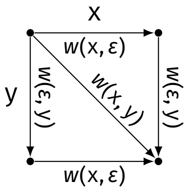


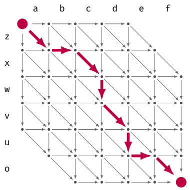
- X and Y consist in $O(k^2)$ pieces of length $O(k^2)$ and with period $O(k)$ each
- Idea: Use (directed) alignment graph AG of X and Y (seaweeds don't work here)
  $\rightsquigarrow$ $\mathrm{ED}^w(X, Y)$ is distance $(0,0) \rightsquigarrow (|X|, |Y|)$
- Idea²: Trim AG to $O(k)$ diagonals $\rightsquigarrow$ $\mathrm{ED}^w_{\leq k}(X, Y)$ is distance $(0,0) \rightsquigarrow (|X|, |Y|)$
- Idea³: Split AG according to structure of X and Y
  Compute all b-to-b dist [Klein05] + stitch together results ((min, +)-product [SMAWK87])



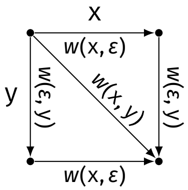


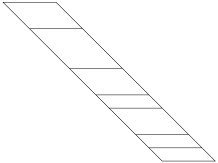
- X and Y consist in $O(k^2)$ pieces of length $O(k^2)$ and with period $O(k)$ each
- Idea: Use (directed) alignment graph AG of X and Y (seaweeds don't work here)
  $\rightsquigarrow$ $\mathrm{ED}^w(X, Y)$ is distance $(0,0) \rightsquigarrow (|X|, |Y|)$
- Idea^2: Trim AG to $O(k)$ diagonals $\rightsquigarrow$ $\mathrm{ED}^w_{\leq k}(X, Y)$ is distance $(0,0) \rightsquigarrow (|X|, |Y|)$
  $\rightsquigarrow$ AG has $O(k^5)$ vertices $\rightsquigarrow$ Dijkstra yields $\tilde{O}(k^5)$ algo
- Idea^3: Split AG according to structure of X and Y
  Compute all b-to-b dist [Klein05] + stitch together results ($(\min, +)$-product [SMAWK87])



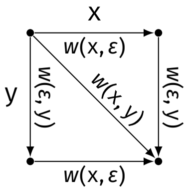


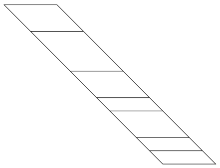
- X and Y consist in $O(k^2)$ pieces of length $O(k^2)$ and with period $O(k)$ each
- Idea: Use (directed) alignment graph AG of X and Y (seaweeds don't work here)
  $\rightsquigarrow \mathrm{ED}^w(X, Y)$ is distance $(0, 0) \rightsquigarrow (|X|, |Y|)$
- Idea^2: Trim AG to $O(k)$ diagonals $\rightsquigarrow \mathrm{ED}^w_{\leq k}(X, Y)$ is distance $(0, 0) \rightsquigarrow (|X|, |Y|)$
- Idea^3: Split AG according to structure of X and Y
  Compute all b-to-b dist [Klein05] + stitch together results ((min, +)-product [SMAWK87])



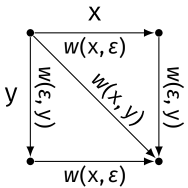


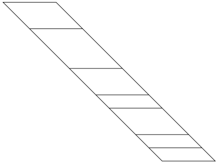
- X and Y consist in $O(k^2)$ pieces of length $O(k^2)$ and with period $O(k)$ each
- Idea: Use (directed) alignment graph AG of X and Y (seaweeds don't work here)
  $\rightsquigarrow$ $\mathrm{ED}^w(X, Y)$ is distance $(0, 0) \rightsquigarrow (|X|, |Y|)$
- Idea^2: Trim AG to $O(k)$ diagonals $\rightsquigarrow$ $\mathrm{ED}^w_{\leq k}(X, Y)$ is distance $(0, 0) \rightsquigarrow (|X|, |Y|)$
- Idea^3: Split AG according to structure of X and Y
  Compute all b-to-b dist [Klein05] + stitch together results ((min, +)-product [SMAWK87])
  $\rightsquigarrow$ $k^2 \cdot \tilde{O}(k^3)$ for periodic pieces + $k^2 \cdot \tilde{O}(k^2)$ for stitching





- X and Y consist in $O(k^2)$ pieces of length $O(k^2)$ and with period $O(k)$ each
- Idea: Use (directed) alignment graph AG of X and Y (seaweeds don't work here)
  $\rightsquigarrow$ $\mathrm{ED}^w(X, Y)$ is distance $(0,0) \rightsquigarrow (|X|, |Y|)$
- Idea²: Trim AG to $O(k)$ diagonals $\rightsquigarrow$ $\mathrm{ED}^w_{\leq k}(X, Y)$ is distance $(0,0) \rightsquigarrow (|X|, |Y|)$
- Idea³: Split AG according to structure of X and Y
  Compute all b-to-b dist [Klein05] + stitch together results ((min, +)-product [SMAWK87])
  for periodic pieces: fast exponentiation;
  $\rightsquigarrow$ $k^2 \cdot \tilde{O}(k^2)$ for periodic pieces + $k^2 \cdot \tilde{O}(k^2)$ for stitching





- X and Y consist in $O(k^2)$ periodic pieces of length $O(k^2)$ and with period $O(k)$ each
- Idea: Use AG, trimmed to $O(k)$ diags $\rightsquigarrow$ $\mathrm{ED}^{w}_{\le k}(X, Y)$ is distance $(0, 0) \rightsquigarrow (|X|, |Y|)$
- Idea[3]: Compute all b-to-b dist for periodic pieces [Klein05] + fast exponentiation;
  stitch together results using min-plus product [SMAWK87]
- Idea[4]: Use Divide-and-Conquer to reduce number of periodic pieces to $O(k)$

  $\rightsquigarrow k \cdot \tilde{O}(k^2)$ for periodic pieces (and padding) + $k \cdot \tilde{O}(k^2)$ for stitching

- X and Y consist in $O(k^2)$ periodic pieces of length $O(k^2)$ and with period $O(k)$ each
- Idea: Use AG, trimmed to $O(k)$ diags $\rightsquigarrow$ $\mathrm{ED}^{w}_{\le k}(X, Y)$ is distance $(0, 0) \rightsquigarrow (|X|, |Y|)$
- Idea^3: Compute all b-to-b dist for periodic pieces [Klein05] + fast exponentiation;
  stitch together results using min-plus product [SMAWK87]
- Idea^4: Use Divide-and-Conquer to reduce number of periodic pieces to $O(k)$
  $\rightsquigarrow$ $k \cdot \tilde{O}(k^2)$ for periodic pieces (and padding) + $k \cdot \tilde{O}(k^2)$ for stitching

- X and Y consist in $O(k)$ pieces of length $O(k^2)$ each
- Idea: Use AG, trimmed to $O(k)$ diags $\rightsquigarrow$ $\mathrm{ED}^{w}_{\le k}(X, Y)$ is distance $(0, 0) \rightsquigarrow (|X|, |Y|)$
- Idea³: Compute all b-to-b dist for periodic pieces [Klein05] + fast exponentiation;
  stitch together results using min-plus product [SMAWK87]
- Idea⁴: Use Divide-and-Conquer to reduce number of periodic pieces to $O(k)$
- Idea⁵: Use tailor-made compressibility measure instead of periodicity
  + (w)ED algorithms for compressed strings
  $\rightsquigarrow \tilde{O}(n + \sqrt{k^3 n})$ time in total

Main Theorem 1 (Upper Bound) ✓

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 2 (Lower Bound)

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

Pattern Matching with Weighted Edits—Next Steps

- Have good algorithm for Bounded Weighted Edit Distance
- (ongoing work) Use structural result and directed alignment graph for $\tilde{O}(|T| + k^4|T|/|P|)$ for PM w/ WE
- Seaweeds don't help,
  but fast multiplication of Monge Matrices still does (at a cost of an extra $O(k)$)
- Still need to solve a lot of extra technical challenges
  (actually obtaining an analogue of [LV89] for verifying starting positions; need to generalize DPM; ...)

Main Results of today

- Structural characterization of strings and PM w/ Edits
- Main tools required for $\tilde{O}(|T| + k^{3.5}|T|/|P|)$ algo for PM w/ E
- Some ideas of how to generalize to the weighted setting

Big Open Questions

- PM w/ Edits in $o(|T| + k^3 \cdot |T|/|P|)$
- PM w/ Weighted Edits in $o(|T| + k^4 \cdot |T|/|P|)$
- Better algorithms known for small integer weights [GK24+], might give better PM algos in that setting

Navigation

Start

Contents

End