The streaming *k*-mismatch problem

Raphaël Clifford **Tomasz Kociumaka** Ely Porat

University of BRISTOL

UNIVERSITY OF WARSAW

Tradition of Excellence Bar-Ilan University

SODA 2019

San Diego, California, January 7, 2019

Pattern matching

Exact pattern matching

Given two strings: a **pattern** P (of length m) and a **text** T (of length n), find all fragments of T **matching** P.

P

b b a a b b b

T

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

Classic algorithms

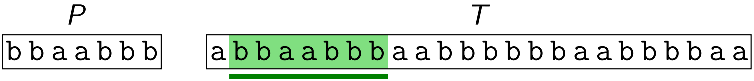
Knuth, Morris, Pratt 1978, SIAM J. Comput.	$\mathcal{O}(n+m)$ time	$\mathcal{O}(m)$ space
Galil, Seiferas 1983, J. Comput. Syst. Sci.	$\mathcal{O}(n+m)$ time	$\mathcal{O}(1)$ space[1]

[1] Does not include read-only random access to P and T.

Pattern matching

Exact pattern matching

Given two strings: a **pattern** P (of length m) and a **text** T (of length n), find all fragments of T **matching** P.



P

b b a a b b b

T

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

Classic algorithms

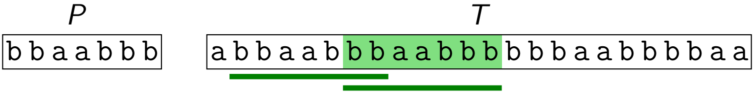
Knuth, Morris, Pratt 1978, SIAM J. Comput.	$\mathcal{O}(n+m)$ time	$\mathcal{O}(m)$ space
Galil, Seiferas 1983, J. Comput. Syst. Sci.	$\mathcal{O}(n+m)$ time	$\mathcal{O}(1)$ space[1]

[1] Does not include read-only random access to P and T.

Pattern matching

Exact pattern matching

Given two strings: a **pattern** P (of length m) and a **text** T (of length n), find all fragments of T **matching** P.



Classic algorithms

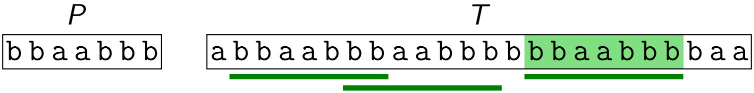
Knuth, Morris, Pratt 1978, SIAM J. Comput.	$\mathcal{O}(n+m)$ time	$\mathcal{O}(m)$ space
Galil, Seiferas 1983, J. Comput. Syst. Sci.	$\mathcal{O}(n+m)$ time	$\mathcal{O}(1)$ space[1]

[1] Does not include read-only random access to P and T.

Pattern matching

Exact pattern matching

Given two strings: a **pattern** P (of length m) and a **text** T (of length n), find all fragments of T **matching** P.



P

b b a a b b b

T

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

Classic algorithms

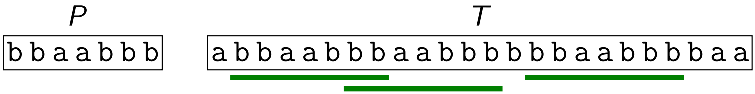
Knuth, Morris, Pratt 1978, SIAM J. Comput.	$\mathcal{O}(n+m)$ time	$\mathcal{O}(m)$ space
Galil, Seiferas 1983, J. Comput. Syst. Sci.	$\mathcal{O}(n+m)$ time	$\mathcal{O}(1)$ space[1]

[1] Does not include read-only random access to P and T.

Pattern matching

Exact pattern matching

Given two strings: a **pattern** P (of length m) and a **text** T (of length n), find all fragments of T **matching** P.



Classic algorithms

Knuth, Morris, Pratt 1978, SIAM J. Comput.	$\mathcal{O}(n+m)$ time	$\mathcal{O}(m)$ space
Galil, Seiferas 1983, J. Comput. Syst. Sci.	$\mathcal{O}(n+m)$ time	$\mathcal{O}(1)$ space[1]

[1] Does not include read-only random access to P and T.

Streaming pattern matching

Data stream model:

- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).

Streaming pattern matching

Data stream model:

- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).

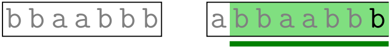
b b a a b b b

a b b a a b b

Streaming pattern matching

Data stream model:

- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).



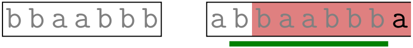
bbaabbb

abbaabbb

Streaming pattern matching

Data stream model:

- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).



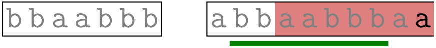
b b a a b b b

a b b a a b b b a

Streaming pattern matching

Data stream model:

- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).



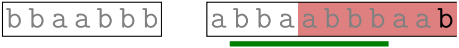
bbaabbb

abbaabbbaa

Streaming pattern matching

Data stream model:

- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).



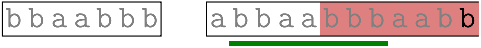
bbaabbb

abbaabbbaab

Streaming pattern matching

Data stream model:

- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).



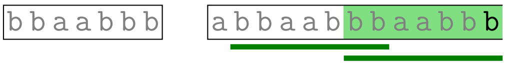
bbaabbb

abbaabbbaabb

Streaming pattern matching

Data stream model:

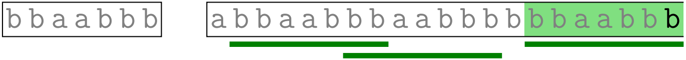
- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).



Streaming pattern matching

Data stream model:

- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).



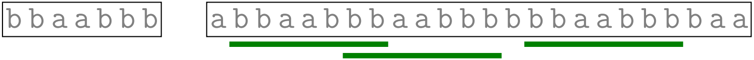
bbaabbb

abbaabbbaabbbbbbaabbb

Streaming pattern matching

Data stream model:

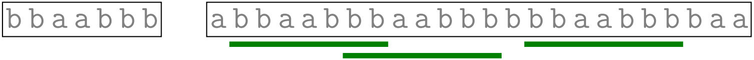
- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).



Streaming pattern matching

Data stream model:

- single sequential scan of the input data,
- online (partial answers after processing each symbol),
- small working space,
- real-time (worst-case per symbol processing time).



Lower bounds

- deterministic: $\Omega(m \log \sigma)$ bits,
- randomized: $\Omega(\log m)$ bits.

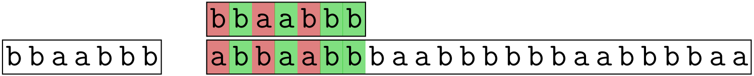
Randomized algorithms

Porat, Porat FOCS 2009	$\mathcal{O}(\log m)$ time	$\mathcal{O}(\log^2 m)$ bits
Breslauer, Galil 2014, ACM Trans. Algorithms	$\mathcal{O}(1)$ time	$\mathcal{O}(\log^2 m)$ bits

Approximate pattern matching

Pattern matching with mismatches

Given a pattern P of length m and a text T of length n, compute the **Hamming distances** between P and all length-m fragments of T.

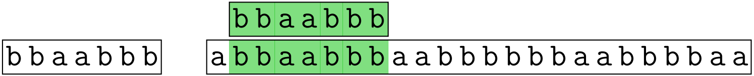


Output: 3

Approximate pattern matching

Pattern matching with mismatches

Given a pattern P of length m and a text T of length n, compute the **Hamming distances** between P and all length-m fragments of T.



b b a a b b b

b b a a b b b

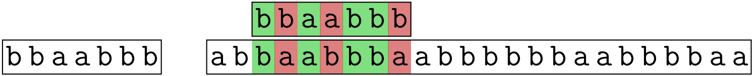
a b b a a b b b a a b b b b b b a a b b b b a a

Output: 3 0

Approximate pattern matching

Pattern matching with mismatches

Given a pattern P of length m and a text T of length n, compute the **Hamming distances** between P and all length-m fragments of T.



b b a a b b b

b b a a b b b

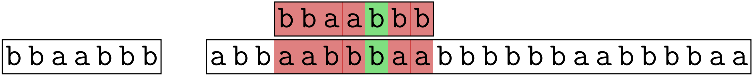
a b b a a b b b a a b b b b b b a a b b b b a a

Output: 3 0 3

Approximate pattern matching

Pattern matching with mismatches

Given a pattern P of length m and a text T of length n, compute the **Hamming distances** between P and all length-m fragments of T.

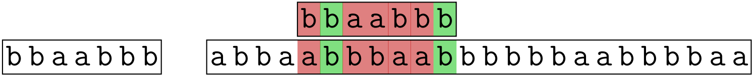


Output: 3 0 3 6

Approximate pattern matching

Pattern matching with mismatches

Given a pattern P of length m and a text T of length n, compute the **Hamming distances** between P and all length-m fragments of T.



b b a a b b b

b b a a b b b

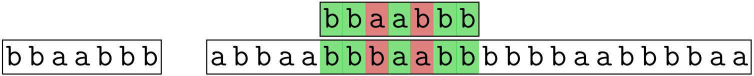
a b b a a b b b a a b b b b b b a a b b b b a a

Output: 3 0 3 6 5

Approximate pattern matching

Pattern matching with mismatches

Given a pattern P of length m and a text T of length n, compute the **Hamming distances** between P and all length-m fragments of T.



Output: 3 0 3 6 5 2

Approximate pattern matching

Pattern matching with mismatches

Given a pattern P of length m and a text T of length n, compute the **Hamming distances** between P and all length-m fragments of T.

```
b b a a b b b     a b b a a b b b a a b b b b b b a a b b b b a a
```

Output: 3 0 3 6 5 2 0 2 4 3 3 4 4 2 0 2 5 5

Approximate pattern matching

Pattern matching with mismatches

Given a pattern P of length m and a text T of length n, compute the **Hamming distances** between P and all length-m fragments of T.

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

Output: 3 0 3 6 5 2 0 2 4 3 3 4 4 2 0 2 5 5

Algorithms

Fischer, Patterson
1973, Complex. Comput. $\mathcal{O}(n\sigma \log m)$ time

Abrahamson
1987, SIAM J. Comput. $\mathcal{O}(n\sqrt{m \log m})$ time

Approximate pattern matching

Pattern matching with mismatches

Given a pattern P of length m and a text T of length n, compute the **Hamming distances** between P and all length-m fragments of T.

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

Output: 3 0 3 6 5 2 0 2 4 3 3 4 4 2 0 2 5 5

Algorithms

Fischer, Patterson
1973, Complex. Comput.
$\mathcal{O}(n\sigma \log m)$ time

Abrahamson
1987, SIAM J. Comput.
$\mathcal{O}(n\sqrt{m \log m})$ time

Lower bound

- no $\mathcal{O}(nm^{0.5-\varepsilon})$-time **combinatorial** algorithms, conditioned on BMM

The k-mismatch problem

Problem

Given a pattern P, a text T, and a **threshold** k, find all fragments of the text T at Hamming distance **at most** k from P (along with the distances).

$\mathbf{k = 3}$

b b a a b b b

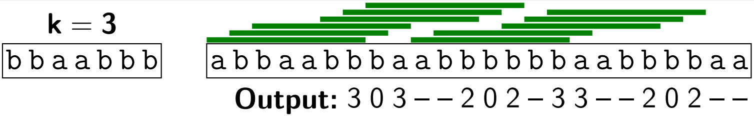
a b b a a b b b a a b b b b b b a a b b b b a a

3 0 3 6 5 **2 0 2** 4 **3 3** 4 4 **2 0 2** 5 5

The k-mismatch problem

Problem

Given a pattern P, a text T, and a **threshold** k, find all fragments of the text T at Hamming distance **at most** k from P (along with the distances).



$\mathbf{k = 3}$

b b a a b b b

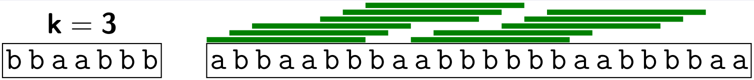
a b b a a b b b a a b b b b b b a a b b b b a a

Output: 3 0 3 – – 2 0 2 – 3 3 – – 2 0 2 – –

The k-mismatch problem

Problem

Given a pattern P, a text T, and a **threshold** k, find all fragments of the text T at Hamming distance **at most** k from P (along with the distances).



$\mathbf{k = 3}$

b b a a b b b

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

Output: 3 0 3 – – 2 0 2 – 3 3 – – 2 0 2 – –

Algorithms

Landau, Vishkin 1986, Theor. Comput. Sci.	$\mathcal{O}(nk)$ time
Amir, Lewenstein, Porat 2004, J. Algorithms	$\mathcal{O}(n\sqrt{k \log k})$ time $\mathcal{O}(n + nk^3 \log k/m)$ time
Clifford et al. SODA 2016	$\widetilde{\mathcal{O}}(n + nk^2/m)$ time
Gawrychowski, Uznański ICALP 2018	$\widetilde{\mathcal{O}}(n + nk/\sqrt{m})$ time **Tight** for combinatorial algorithms (from BMM).

The k-mismatch problem: online and streaming algorithms

Algorithms	Time per symbol	Space in bits

Lower bounds

The k-mismatch problem: online and streaming algorithms

Algorithms	Time per symbol	Space in bits	
Clifford et al. 2011, Inf. Comput. (CPM 2008)	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\mathcal{O}(m \log m)$	deterministic

Lower bounds

The *k*-mismatch problem: online and streaming algorithms

Algorithms	Time per symbol	Space in bits	
Clifford et al. 2011, Inf. Comput. (CPM 2008)	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\mathcal{O}(m \log m)$	deterministic

Lower bounds			
Folklore		$\Omega(m \log \sigma)$	deterministic

The k-mismatch problem: online and streaming algorithms

Algorithms	Time per symbol	Space in bits	
Clifford et al. 2011, Inf. Comput. (CPM 2008)	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\mathcal{O}(m \log m)$	deterministic
Lower bounds			
Folklore		$\Omega(m \log \sigma)$	deterministic
Gawrychowski, Uznański 2018, personal communication	$\Omega(k^{0.5-\varepsilon})$		combinatorial

The k-mismatch problem: online and streaming algorithms

Algorithms	Time per symbol	Space in bits	
Clifford et al. 2011, Inf. Comput. (CPM 2008)	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\mathcal{O}(m \log m)$	deterministic
Porat, Porat FOCS 2009	$\widetilde{\mathcal{O}}(k^2)$	$\widetilde{\mathcal{O}}(k^3)$	randomized

Lower bounds			
Folklore		$\Omega(m \log \sigma)$	deterministic
Gawrychowski, Uznański 2018, personal communication	$\Omega(k^{0.5-\varepsilon})$		combinatorial

The k-mismatch problem: online and streaming algorithms

Algorithms	Time per symbol	Space in bits	
Clifford et al. 2011, Inf. Comput. (CPM 2008)	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\mathcal{O}(m \log m)$	deterministic
Porat, Porat FOCS 2009	$\widetilde{\mathcal{O}}(k^2)$	$\widetilde{\mathcal{O}}(k^3)$	randomized

Lower bounds			
Folklore		$\Omega(m \log \sigma)$	deterministic
Huang et al. 2006, Inf. Process. Lett.		$\Omega(k + \log n)$	randomized
Gawrychowski, Uznański 2018, personal communication	$\Omega(k^{0.5-\varepsilon})$		combinatorial

The k-mismatch problem: online and streaming algorithms

Algorithms	Time per symbol	Space in bits	
Clifford et al. 2011, Inf. Comput. (CPM 2008)	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\mathcal{O}(m \log m)$	deterministic
Porat, Porat FOCS 2009	$\widetilde{\mathcal{O}}(k^2)$	$\widetilde{\mathcal{O}}(k^3)$	randomized
Clifford et al. SODA 2016	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\widetilde{\mathcal{O}}(k^2)$	randomized
Golan, Kopelowitz, Porat ICALP 2018	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(k)$	randomized

Lower bounds			
Folklore		$\Omega(m \log \sigma)$	deterministic
Huang et al. 2006, Inf. Process. Lett.		$\Omega(k + \log n)$	randomized
Gawrychowski, Uznański 2018, personal communication	$\Omega(k^{0.5-\varepsilon})$		combinatorial

The k-mismatch problem: online and streaming algorithms

Algorithms	Time per symbol	Space in bits	
Clifford et al. 2011, Inf. Comput. (CPM 2008)	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\mathcal{O}(m \log m)$	deterministic
Porat, Porat FOCS 2009	$\widetilde{\mathcal{O}}(k^2)$	$\widetilde{\mathcal{O}}(k^3)$	randomized
Clifford et al. SODA 2016	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\widetilde{\mathcal{O}}(k^2)$	randomized
Golan, Kopelowitz, Porat ICALP 2018	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(k)$	randomized

Lower bounds			
Folklore		$\Omega(m \log \sigma)$	deterministic
Huang et al. 2006, Inf. Process. Lett.		$\Omega(k + \log n)$	randomized
Gawrychowski, Uznański 2018, personal communication	$\Omega(k^{0.5-\varepsilon})$		combinatorial

The k-mismatch problem: online and streaming algorithms

Algorithms	Time per symbol	Space in bits	
Clifford et al. 2011, Inf. Comput. (CPM 2008)	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\mathcal{O}(m \log m)$	deterministic
Porat, Porat FOCS 2009	$\widetilde{\mathcal{O}}(k^2)$	$\widetilde{\mathcal{O}}(k^3)$	randomized
Clifford et al. SODA 2016	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\widetilde{\mathcal{O}}(k^2)$	randomized
Golan, Kopelowitz, Porat ICALP 2018	$\widetilde{\mathcal{O}}(k)$	$\widetilde{\mathcal{O}}(k)$	randomized
This work SODA 2019	$\widetilde{\mathcal{O}}(\sqrt{k})$	$\widetilde{\mathcal{O}}(k)$	randomized
Lower bounds			
Folklore		$\Omega(m \log \sigma)$	deterministic
Huang et al. 2006, Inf. Process. Lett.		$\Omega(k + \log n)$	randomized
Gawrychowski, Uznański 2018, personal communication	$\Omega(k^{0.5-\varepsilon})$		combinatorial

Our main result

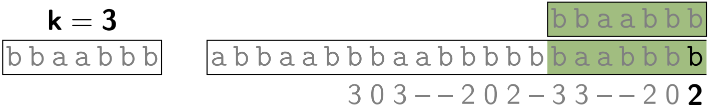
Theorem (This work)

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ *bits of space and takes* $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ *time per symbol.*

Our main result

Theorem (This work)

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ bits of space and takes $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ time per symbol.



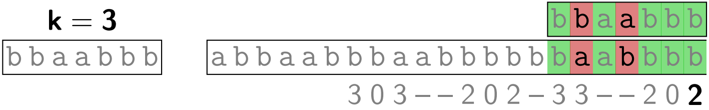
Extra features of the new algorithm:

- For each reported occurrence, the **mismatch information** can be computed on demand in $\mathcal{O}(k)$ time.

Our main result

Theorem (This work)

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ bits of space and takes $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ time per symbol.



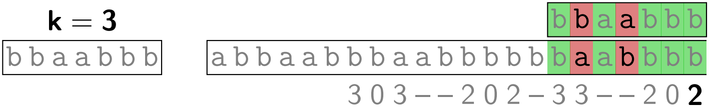
Extra features of the new algorithm:

- For each reported occurrence, the **mismatch information** can be computed on demand in $\mathcal{O}(k)$ time.

Our main result

Theorem (This work)

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ bits of space and takes $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ time per symbol.



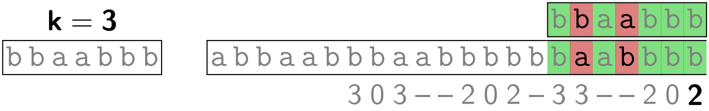
Extra features of the new algorithm:

- For each reported occurrence, the **mismatch information** can be computed on demand in $\mathcal{O}(k)$ time.
  The only previous streaming algorithm computing mismatch information:
  Radoszewski, Starikovskaya (DCC 2017): $\widetilde{\mathcal{O}}(k)$ time per symbol, $\widetilde{\mathcal{O}}(k^2)$ space.

Our main result

Theorem (This work)

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ bits of space and takes $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ time per symbol.



$\mathbf{k = 3}$

b b a a b b b

b b a a b b b

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

3 0 3 − − 2 0 2 − 3 3 − − 2 0 **2**

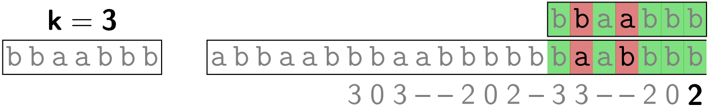
Extra features of the new algorithm:

- For each reported occurrence, the **mismatch information** can be computed on demand in $\mathcal{O}(k)$ time.
  The only previous streaming algorithm computing mismatch information: Radoszewski, Starikovskaya (DCC 2017): $\widetilde{\mathcal{O}}(k)$ time per symbol, $\widetilde{\mathcal{O}}(k^2)$ space.
- Pattern preprocessing under the same bounds on space and time.

Our main result

Theorem (This work)

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ *bits of space and takes* $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ *time per symbol.*



Extra features of the new algorithm:

- For each reported occurrence, the **mismatch information** can be computed on demand in $\mathcal{O}(k)$ time.
  The only previous streaming algorithm computing mismatch information: Radoszewski, Starikovskaya (DCC 2017): $\widetilde{\mathcal{O}}(k)$ time per symbol, $\widetilde{\mathcal{O}}(k^2)$ space.
- Pattern preprocessing under the same bounds on space and time.
  All previous algorithms require non-streaming preprocessing.

Outline of the talk

Introduction

Exact streaming pattern matching

Our streaming k-mismatch algorithm

Conclusions and open problems

Outline of the talk

Introduction

Exact streaming pattern matching

Our streaming k-mismatch algorithm

Conclusions and open problems

Online pattern matching in $\mathcal{O}(\log n)$ bits[1]

Karp and Rabin (1987, IBM J. Res. Dev.)

Karp–Rabin fingerprints

Assign $\mathcal{O}(\log m)$-bit integer **fingerprints** $\Psi(\cdot)$ to strings of length up to m so that if $X \neq Y$, then $\Pr[\Psi(X) = \Psi(Y)] \leq m^{-\Theta(1)}$.

[1] Plus read-only access to the text.

Online pattern matching in $\mathcal{O}(\log n)$ bits[1]

Karp and Rabin (1987, IBM J. Res. Dev.)

Karp–Rabin fingerprints

Assign $\mathcal{O}(\log m)$-bit integer **fingerprints** $\Psi(\cdot)$ to strings of length up to m so that if $X \neq Y$, then $\Pr[\Psi(X) = \Psi(Y)] \leq m^{-\Theta(1)}$.

Rolling fingerprints:
Any of $\Psi(X), \Psi(Y), \Psi(XY)$ can be retrieved from the other two.

[1] Plus read-only access to the text.

Online pattern matching in $\mathcal{O}(\log n)$ bits[1]

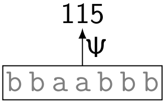
Karp and Rabin (1987, IBM J. Res. Dev.)

Karp–Rabin fingerprints

Assign $\mathcal{O}(\log m)$-bit integer **fingerprints** $\Psi(\cdot)$ to strings of length up to m so that if $X \neq Y$, then $\Pr[\Psi(X) = \Psi(Y)] \leq m^{-\Theta(1)}$.

Rolling fingerprints:

Any of $\Psi(X), \Psi(Y), \Psi(XY)$ can be retrieved from the other two.



115

Ψ

b b a a b b b

[1] Plus read-only access to the text.

Online pattern matching in $\mathcal{O}(\log n)$ bits[1]

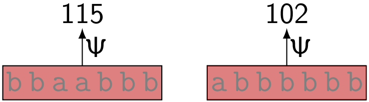
Karp and Rabin (1987, IBM J. Res. Dev.)

Karp–Rabin fingerprints

Assign $\mathcal{O}(\log m)$-bit integer **fingerprints** $\Psi(\cdot)$ to strings of length up to m so that if $X \neq Y$, then $\Pr[\Psi(X) = \Psi(Y)] \leq m^{-\Theta(1)}$.

Rolling fingerprints:

Any of $\Psi(X), \Psi(Y), \Psi(XY)$ can be retrieved from the other two.



[1] Plus read-only access to the text.

Online pattern matching in $\mathcal{O}(\log n)$ bits[1]

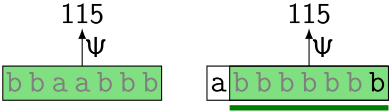
Karp and Rabin (1987, IBM J. Res. Dev.)

Karp–Rabin fingerprints

Assign $\mathcal{O}(\log m)$-bit integer **fingerprints** $\Psi(\cdot)$ to strings of length up to m so that if $X \neq Y$, then $\Pr[\Psi(X) = \Psi(Y)] \leq m^{-\Theta(1)}$.

Rolling fingerprints:

Any of $\Psi(X), \Psi(Y), \Psi(XY)$ can be retrieved from the other two.



[1] Plus read-only access to the text.

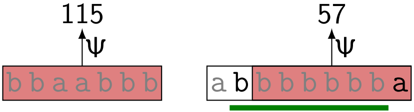
Online pattern matching in $\mathcal{O}(\log n)$ bits[1]

Karp and Rabin (1987, IBM J. Res. Dev.)

Karp–Rabin fingerprints

Assign $\mathcal{O}(\log m)$-bit integer **fingerprints** $\Psi(\cdot)$ to strings of length up to m so that if $X \neq Y$, then $\Pr[\Psi(X) = \Psi(Y)] \leq m^{-\Theta(1)}$.

Rolling fingerprints:
Any of $\Psi(X), \Psi(Y), \Psi(XY)$ can be retrieved from the other two.



[1] Plus read-only access to the text.

Online pattern matching in $\mathcal{O}(\log n)$ bits[1]

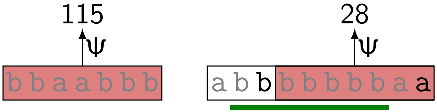
Karp and Rabin (1987, IBM J. Res. Dev.)

Karp–Rabin fingerprints

Assign $\mathcal{O}(\log m)$-bit integer **fingerprints** $\Psi(\cdot)$ to strings of length up to m so that if $X \neq Y$, then $\Pr[\Psi(X) = \Psi(Y)] \leq m^{-\Theta(1)}$.

Rolling fingerprints:

Any of $\Psi(X), \Psi(Y), \Psi(XY)$ can be retrieved from the other two.



[1] Plus read-only access to the text.

Online pattern matching in $\mathcal{O}(\log n)$ bits[1]

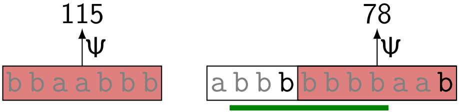
Karp and Rabin (1987, IBM J. Res. Dev.)

Karp–Rabin fingerprints

Assign $\mathcal{O}(\log m)$-bit integer **fingerprints** $\Psi(\cdot)$ to strings of length up to m so that if $X \neq Y$, then $\Pr[\Psi(X) = \Psi(Y)] \leq m^{-\Theta(1)}$.

Rolling fingerprints:

Any of $\Psi(X), \Psi(Y), \Psi(XY)$ can be retrieved from the other two.



[1] Plus read-only access to the text.

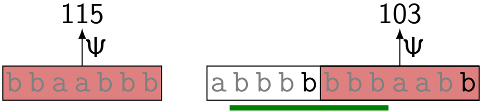
Online pattern matching in $\mathcal{O}(\log n)$ bits[1]

Karp and Rabin (1987, IBM J. Res. Dev.)

Karp–Rabin fingerprints

Assign $\mathcal{O}(\log m)$-bit integer **fingerprints** $\Psi(\cdot)$ to strings of length up to m so that if $X \neq Y$, then $\Pr[\Psi(X) = \Psi(Y)] \leq m^{-\Theta(1)}$.

Rolling fingerprints:
Any of $\Psi(X), \Psi(Y), \Psi(XY)$ can be retrieved from the other two.



[1] Plus read-only access to the text.

Online pattern matching in $\mathcal{O}(\log n)$ bits[1]

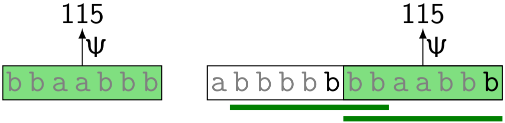
Karp and Rabin (1987, IBM J. Res. Dev.)

Karp–Rabin fingerprints

Assign $\mathcal{O}(\log m)$-bit integer **fingerprints** $\Psi(\cdot)$ to strings of length up to m so that if $X \neq Y$, then $\Pr[\Psi(X) = \Psi(Y)] \leq m^{-\Theta(1)}$.

Rolling fingerprints:

Any of $\Psi(X), \Psi(Y), \Psi(XY)$ can be retrieved from the other two.



[1] Plus read-only access to the text.

Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

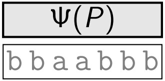
How to avoid accessing this character?

Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?



$\Psi(P)$

b b a a b b b

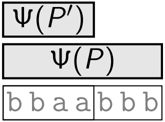
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.



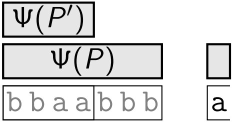
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\mathinner{.\,.}\lceil m/2\rceil]$.
2. Maintain $\Psi(T[1\mathinner{.\,.}i])$.



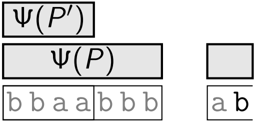
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\,..\,i])$.



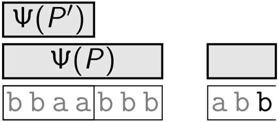
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\,..\,i])$.



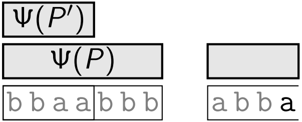
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\,..\,i])$.



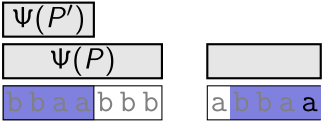
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\,..\,i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\,..\,j-1])$.



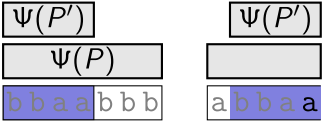
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1 \mathinner{.\,.} \lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1 \mathinner{.\,.} i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1 \mathinner{.\,.} j-1])$.



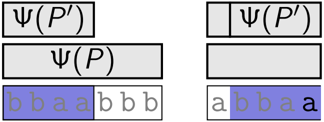
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\,..\,i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\,..\,j-1])$.



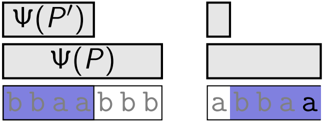
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\mathinner{.\,.}\lceil m/2\rceil]$.
2. Maintain $\Psi(T[1\mathinner{.\,.}i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\mathinner{.\,.}j-1])$.



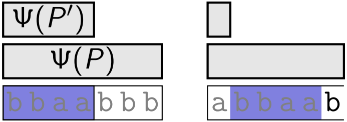
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\mathinner{.\,.}\lceil m/2\rceil]$.
2. Maintain $\Psi(T[1\mathinner{.\,.}i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\mathinner{.\,.}j-1])$.



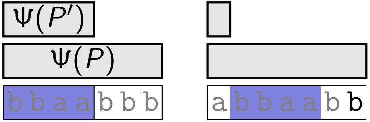
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\,..\,i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\,..\,j-1])$.



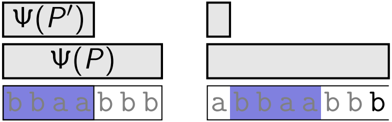
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\,..\,i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\,..\,j-1])$.
4. Combine $\Psi(T[1\,..\,j-1])$ with $\Psi(T[1\,..\,i])$ to check if $P = T[j\,..\,i]$.



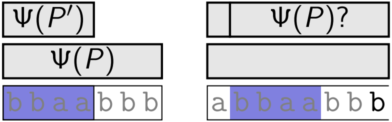
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\mathinner{.\,.}\lceil m/2\rceil]$.
2. Maintain $\Psi(T[1\mathinner{.\,.}i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\mathinner{.\,.}j-1])$.
4. Combine $\Psi(T[1\mathinner{.\,.}j-1])$ with $\Psi(T[1\mathinner{.\,.}i])$ to check if $P = T[j\mathinner{.\,.}i]$.



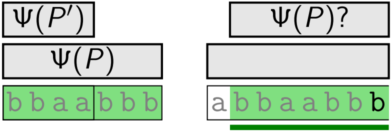
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\,..\,i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\,..\,j-1])$.
4. Combine $\Psi(T[1\,..\,j-1])$ with $\Psi(T[1\,..\,i])$ to check if $P = T[j\,..\,i]$.



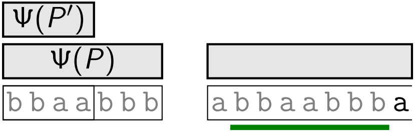
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\,..\,i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\,..\,j-1])$.
4. Combine $\Psi(T[1\,..\,j-1])$ with $\Psi(T[1\,..\,i])$ to check if $P = T[j\,..\,i]$.



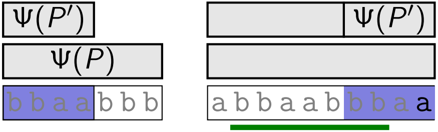
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\,..\,\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\,..\,i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\,..\,j-1])$.
4. Combine $\Psi(T[1\,..\,j-1])$ with $\Psi(T[1\,..\,i])$ to check if $P = T[j\,..\,i]$.



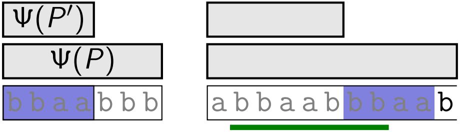
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\mathinner{.\,.}\lceil m/2 \rceil]$.
2. Maintain $\Psi(T[1\mathinner{.\,.}i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\mathinner{.\,.}j-1])$.
4. Combine $\Psi(T[1\mathinner{.\,.}j-1])$ with $\Psi(T[1\mathinner{.\,.}i])$ to check if $P = T[j\mathinner{.\,.}i]$.



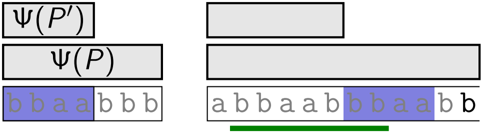
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\mathinner{.\,.}\lceil m/2\rceil]$.
2. Maintain $\Psi(T[1\mathinner{.\,.}i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\mathinner{.\,.}j-1])$.
4. Combine $\Psi(T[1\mathinner{.\,.}j-1])$ with $\Psi(T[1\mathinner{.\,.}i])$ to check if $P = T[j\mathinner{.\,.}i]$.



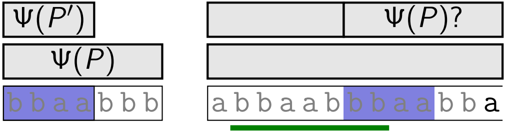
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\mathinner{.\,.}\lceil m/2\rceil]$.
2. Maintain $\Psi(T[1\mathinner{.\,.}i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\mathinner{.\,.}j-1])$.
4. Combine $\Psi(T[1\mathinner{.\,.}j-1])$ with $\Psi(T[1\mathinner{.\,.}i])$ to check if $P = T[j\mathinner{.\,.}i]$.



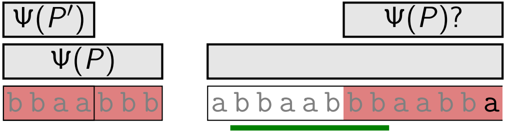
Streaming pattern matching in $\mathcal{O}(\log^2 n)$ bits

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Issue: Rabin–Karp algorithm needs $T[i-m]$ to process $T[i]$.

How to avoid accessing this character?

1. Recursively look for the occurrences of $P' := P[1\mathinner{.\,.}\lceil m/2\rceil]$.
2. Maintain $\Psi(T[1\mathinner{.\,.}i])$.
3. If P' is detected at position j, retrieve and store $\Psi(T[1\mathinner{.\,.}j-1])$.
4. Combine $\Psi(T[1\mathinner{.\,.}j-1])$ with $\Psi(T[1\mathinner{.\,.}i])$ to check if $P = T[j\mathinner{.\,.}i]$.



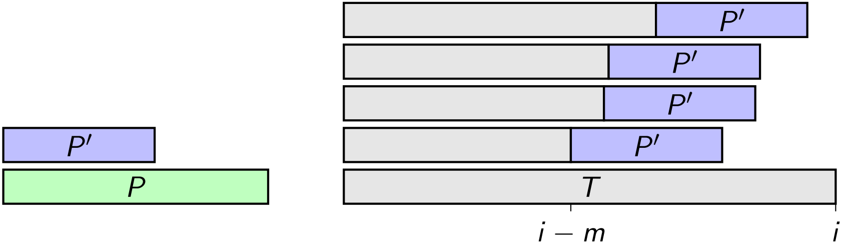
Structure of the viable occurrences of P'

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Viable occurrences of P'

Occurrences of P' in T starting at positions $j \in \{i-m, \ldots, i-|P'|\}$.

- For each viable occurrence, we need to store $\Psi(T[1\mathinner{.\,.}j-1])$.
- There can be $\Theta(m)$ viable occurrences...



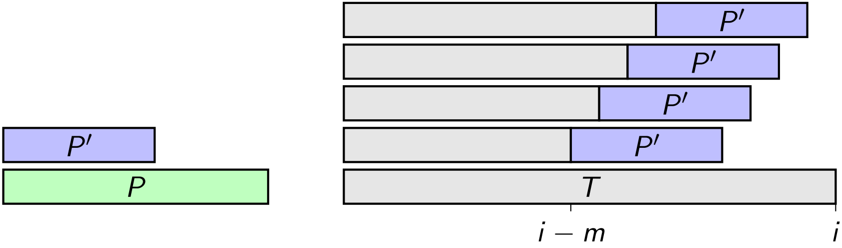
Structure of the viable occurrences of P'

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Viable occurrences of P'

Occurrences of P' in T starting at positions $j \in \{i-m, \ldots, i-|P'|\}$.

- For each viable occurrence, we need to store $\Psi(T[1\,..\,j-1])$.
- There can be $\Theta(m)$ viable occurrences…
  but their starting positions form an arithmetic progression.



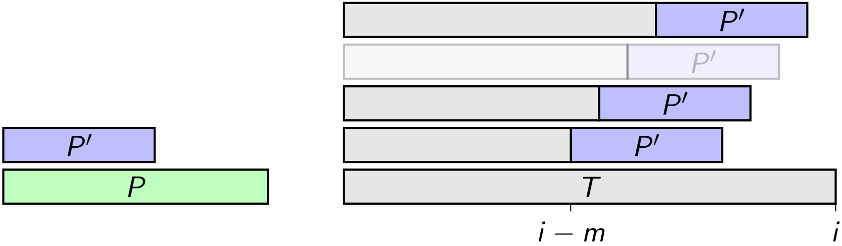
Structure of the viable occurrences of P'

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Viable occurrences of P'

Occurrences of P' in T starting at positions $j \in \{i-m, \ldots, i-|P'|\}$.

- For each viable occurrence, we need to store $\Psi(T[1\,..\,j-1])$.
- There can be $\Theta(m)$ viable occurrences… but their starting positions form an arithmetic progression.
- We store only the first two fingerprints and the last one…



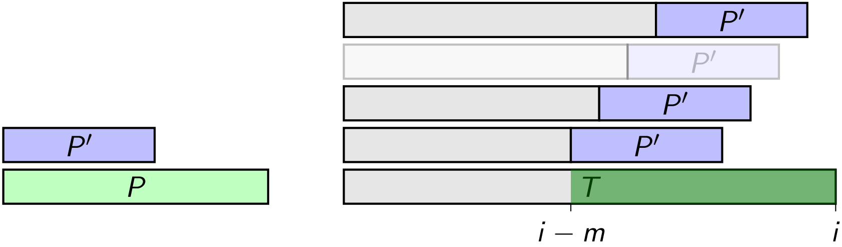
Structure of the viable occurrences of P'

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Viable occurrences of P'

Occurrences of P' in T starting at positions $j \in \{i-m, \ldots, i-|P'|\}$.

- For each viable occurrence, we need to store $\Psi(T[1\,..\,j-1])$.
- There can be $\Theta(m)$ viable occurrences...
  but their starting positions form an arithmetic progression.
- We store only the first two fingerprints and the last one...
  and update the representation when necessary.



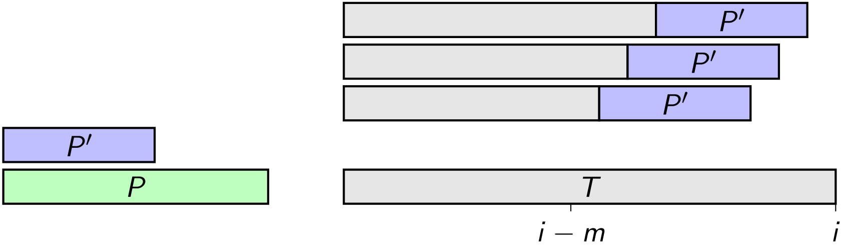
Structure of the viable occurrences of P'

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Viable occurrences of P'

Occurrences of P' in T starting at positions $j \in \{i-m, \ldots, i-|P'|\}$.

- For each viable occurrence, we need to store $\Psi(T[1\,..\,j-1])$.
- There can be $\Theta(m)$ viable occurrences...
  but their starting positions form an arithmetic progression.
- We store only the first two fingerprints and the last one...
  and update the representation when necessary.



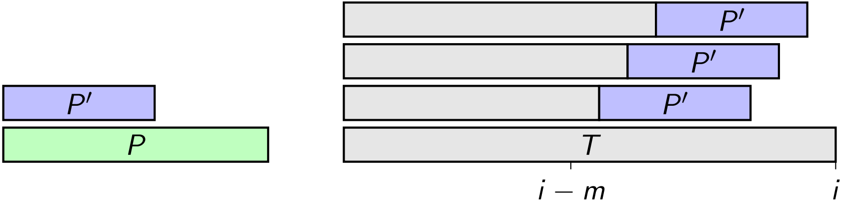
Structure of the viable occurrences of P'

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Viable occurrences of P'

Occurrences of P' in T starting at positions $j \in \{i-m, \ldots, i-|P'|\}$.

- For each viable occurrence, we need to store $\Psi(T[1\,..\,j-1])$.
- There can be $\Theta(m)$ viable occurrences...
  but their starting positions form an arithmetic progression.
- We store only the first two fingerprints and the last one...
  and update the representation when necessary.



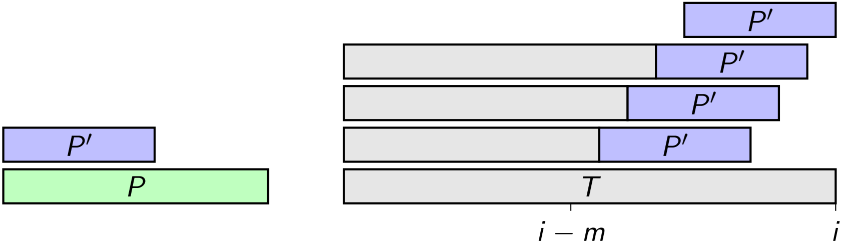
Structure of the viable occurrences of P'

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Viable occurrences of P'

Occurrences of P' in T starting at positions $j \in \{i-m, \ldots, i-|P'|\}$.

- For each viable occurrence, we need to store $\Psi(T[1\,..\,j-1])$.
- There can be $\Theta(m)$ viable occurrences...
  but their starting positions form an arithmetic progression.
- We store only the first two fingerprints and the last one...
  and update the representation when necessary.



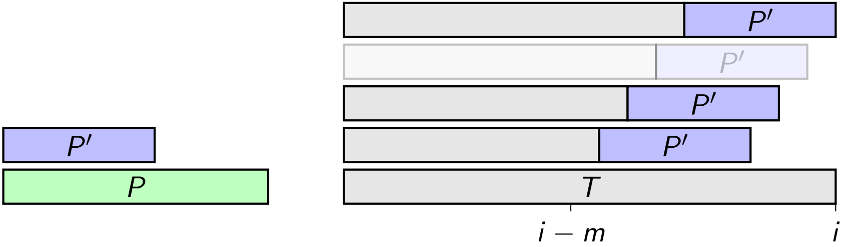
Structure of the viable occurrences of P'

Porat and Porat (FOCS 2009), Breslauer and Galil (2014, ACM Trans. Algorithms)

Viable occurrences of P'

Occurrences of P' in T starting at positions $j \in \{i-m, \ldots, i-|P'|\}$.

- For each viable occurrence, we need to store $\Psi(T[1\,..\,j-1])$.
- There can be $\Theta(m)$ viable occurrences...
  but their starting positions form an arithmetic progression.
- We store only the first two fingerprints and the last one...
  and update the representation when necessary.



Outline of the talk

Introduction

Exact streaming pattern matching

Our streaming k-mismatch algorithm

Conclusions and open problems

Contribution #1: Rolling k-mismatch sketches

The k-mismatch sketches

A **sketch function** sk_k mapping words X, $|X| \le n$, to $\mathcal{O}(k \log n)$-bit values $\mathrm{sk}_k(X)$ designed so that $\mathrm{sk}_k(X)$ and $\mathrm{sk}_k(Y)$ are sufficient to:

- decide whether $\mathrm{HD}(X, Y) \le k$

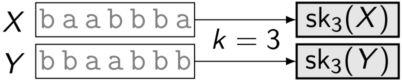
X `baabbba`
Y `bbaabbb`

Contribution #1: Rolling k-mismatch sketches

The k-mismatch sketches

A **sketch function** sk_k mapping words X, $|X| \le n$, to $\mathcal{O}(k \log n)$-bit values $\mathrm{sk}_k(X)$ designed so that $\mathrm{sk}_k(X)$ and $\mathrm{sk}_k(Y)$ are sufficient to:

- decide whether $\mathrm{HD}(X, Y) \le k$

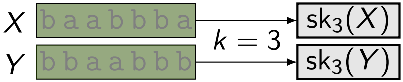


Contribution #1: Rolling k-mismatch sketches

The k-mismatch sketches

A **sketch function** sk_k mapping words X, $|X| \le n$, to $\mathcal{O}(k \log n)$-bit values $\mathsf{sk}_k(X)$ designed so that $\mathsf{sk}_k(X)$ and $\mathsf{sk}_k(Y)$ are sufficient to:

- decide whether $\mathrm{HD}(X, Y) \le k$

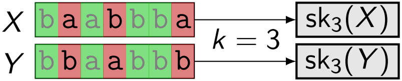


Contribution #1: Rolling k-mismatch sketches

The k-mismatch sketches

A **sketch function** sk_k mapping words X, $|X| \le n$, to $\mathcal{O}(k \log n)$-bit values $\mathrm{sk}_k(X)$ designed so that $\mathrm{sk}_k(X)$ and $\mathrm{sk}_k(Y)$ are sufficient to:

- decide whether $\mathrm{HD}(X, Y) \le k$, and
- retrieve the mismatch information if $\mathrm{HD}(X, Y) \le k$,



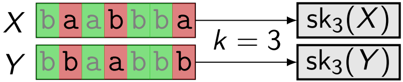
Contribution #1: Rolling k-mismatch sketches

The k-mismatch sketches

A **sketch function** sk_k mapping words X, $|X| \le n$, to $\mathcal{O}(k \log n)$-bit values $\mathrm{sk}_k(X)$ designed so that $\mathrm{sk}_k(X)$ and $\mathrm{sk}_k(Y)$ are sufficient to:

- decide whether $\mathrm{HD}(X, Y) \le k$, and
- retrieve the mismatch information if $\mathrm{HD}(X, Y) \le k$,

both in $\widetilde{\mathcal{O}}(k)$ time.



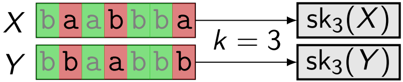
Contribution #1: Rolling k-mismatch sketches

The k-mismatch sketches

A **sketch function** sk_k mapping words X, $|X| \le n$, to $\mathcal{O}(k \log n)$-bit values $\mathsf{sk}_k(X)$ designed so that $\mathsf{sk}_k(X)$ and $\mathsf{sk}_k(Y)$ are sufficient to:

- decide whether $\mathrm{HD}(X, Y) \le k$, and
- retrieve the mismatch information if $\mathrm{HD}(X, Y) \le k$,

both in $\widetilde{\mathcal{O}}(k)$ time.



X b a a b b b a $\to$ $\mathsf{sk}_3(X)$

$k = 3$

Y b b a a b b b $\to$ $\mathsf{sk}_3(Y)$

Manipulation in $\widetilde{\mathcal{O}}(k)$ time:

- concatenation,
- prefix and suffix removal,
- appending $\mathcal{O}(k)$ chars,
- $\mathcal{O}(k)$ substitutions.

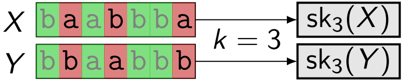
Contribution #1: Rolling k-mismatch sketches

The k-mismatch sketches

A **sketch function** sk_k mapping words X, $|X| \le n$, to $\mathcal{O}(k \log n)$-bit values $\mathrm{sk}_k(X)$ designed so that $\mathrm{sk}_k(X)$ and $\mathrm{sk}_k(Y)$ are sufficient to:

- decide whether $\mathrm{HD}(X, Y) \le k$, and
- retrieve the mismatch information if $\mathrm{HD}(X, Y) \le k$,

both in $\widetilde{\mathcal{O}}(k)$ time.



X b a a b b b a

Y b b a a b b b

$k = 3$

$\mathrm{sk}_3(X)$

$\mathrm{sk}_3(Y)$

Manipulation in $\widetilde{\mathcal{O}}(k)$ time:

- concatenation,
- prefix and suffix removal,
- appending $\mathcal{O}(k)$ chars,
- $\mathcal{O}(k)$ substitutions.

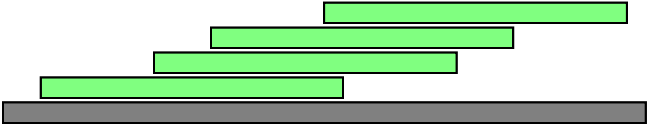
Techniques:

- Reed–Solomon error correcting codes,
- Karp–Rabin fingerprints,
- polynomial factorization, evaluation, and interpolation.

Contribution #2: Encoding viable k-mismatch occurrences

Theorem

The k-mismatch occurrences of a length-m pattern in a length-$2m$ text, each with the mismatch information (MI), can be encoded in $\widetilde{\mathcal{O}}(k)$ bits.

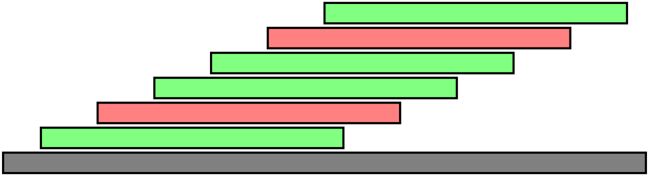


- The starting positions do not form an arithmetic progression…

Contribution #2: Encoding viable k-mismatch occurrences

Theorem

The k-mismatch occurrences of a length-m pattern in a length-$2m$ text, each with the mismatch information (MI), can be encoded in $\widetilde{\mathcal{O}}(k)$ bits.

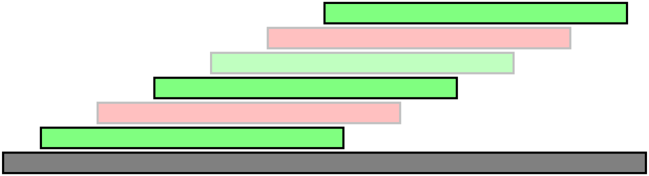


- The starting positions do not form an arithmetic progression…
  but we still consider the smallest progression containing all of them.

Contribution #2: Encoding viable k-mismatch occurrences

Theorem

The k-mismatch occurrences of a length-m pattern in a length-$2m$ text, each with the mismatch information (MI), can be encoded in $\widetilde{\mathcal{O}}(k)$ bits.

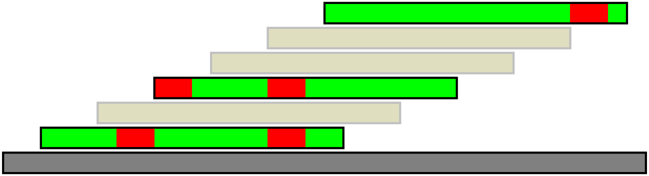


- The starting positions do not form an arithmetic progression...
  but we still consider the smallest progression containing all of them.
- This progression is spanned by $\mathcal{O}(\log m)$ k-mismatch occurrences.

Contribution #2: Encoding viable k-mismatch occurrences

Theorem

The k-mismatch occurrences of a length-m pattern in a length-$2m$ text, each with the mismatch information (MI), can be encoded in $\widetilde{\mathcal{O}}(k)$ bits.

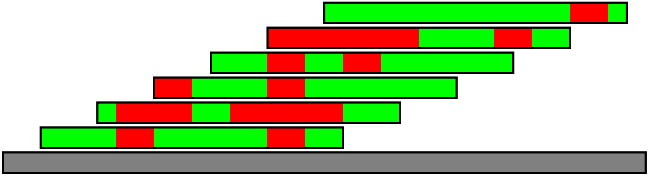


- The starting positions do not form an arithmetic progression…
  but we still consider the smallest progression containing all of them.
- This progression is spanned by $\mathcal{O}(\log m)$ k-mismatch occurrences.
- Their MI encodes the MI for all alignments in the progression.

Contribution #2: Encoding viable k-mismatch occurrences

Theorem

The k-mismatch occurrences of a length-m pattern in a length-2m text, each with the mismatch information (MI), can be encoded in $\widetilde{\mathcal{O}}(k)$ bits.

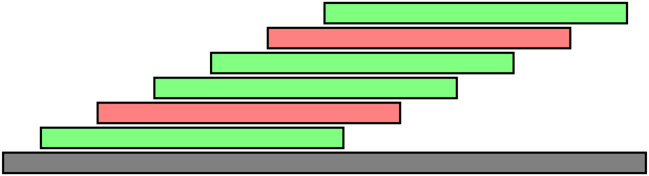


- The starting positions do not form an arithmetic progression...
  but we still consider the smallest progression containing all of them.
- This progression is spanned by $\mathcal{O}(\log m)$ k-mismatch occurrences.
- Their MI encodes the MI for all alignments in the progression.

Contribution #2: Encoding viable k-mismatch occurrences

Theorem

The k-mismatch occurrences of a length-m pattern in a length-$2m$ text, each with the mismatch information (MI), can be encoded in $\widetilde{\mathcal{O}}(k)$ bits.

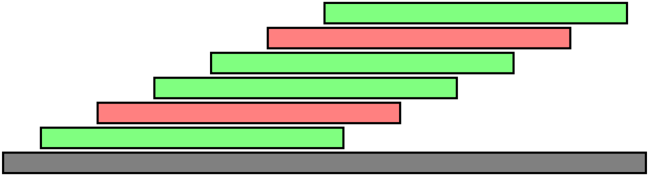


- The starting positions do not form an arithmetic progression...
  but we still consider the smallest progression containing all of them.
- This progression is spanned by $\mathcal{O}(\log m)$ k-mismatch occurrences.
- Their MI encodes the MI for all alignments in the progression.

Contribution #2: Encoding viable k-mismatch occurrences

Theorem

The k-mismatch occurrences of a length-m pattern in a length-$2m$ text, each with the mismatch information (MI), can be encoded in $\widetilde{\mathcal{O}}(k)$ bits.



- The starting positions do not form an arithmetic progression…
  but we still consider the smallest progression containing all of them.
- This progression is spanned by $\mathcal{O}(\log m)$ k-mismatch occurrences.
- Their MI encodes the MI for all alignments in the progression.

Consequence

A k-mismatch streaming algorithm with $\widetilde{\mathcal{O}}(k)$ space and time per symbol.

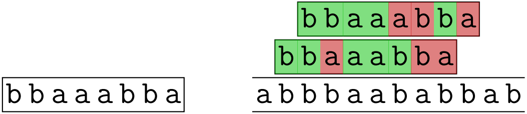
Contribution #3: New solution for nearly periodic patterns

Bottleneck: manipulation of sketches at every position.

Contribution #3: New solution for nearly periodic patterns

Bottleneck: manipulation of sketches at every position.

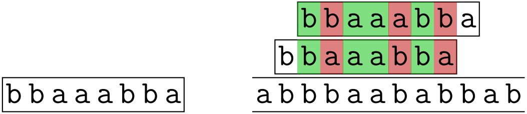
Hope: sketches needed only when a viable occurrence is processed.



Contribution #3: New solution for nearly periodic patterns

Bottleneck: manipulation of sketches at every position.

Hope: sketches needed only when a viable occurrence is processed.



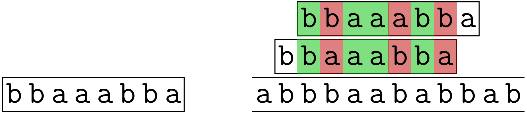
Approximate period

An integer p is a d-**period** of P if $\mathrm{HD}(P[1\mathinner{.\,.}m-p], P[p+1\mathinner{.\,.}m]) \le d$.

Contribution #3: New solution for nearly periodic patterns

Bottleneck: manipulation of sketches at every position.

Hope: sketches needed only when a viable occurrence is processed.



Approximate period

An integer p is a d-**period** of P if $\mathrm{HD}(P[1\,..\,m-p], P[p+1\,..\,m]) \le d$.

Improved specialized algorithm

A **deterministic** streaming k-mismatches algorithm for patterns P with an $\mathcal{O}(k)$-period $\mathcal{O}(k)$. Complexity: $\widetilde{\mathcal{O}}(k)$ bits and $\widetilde{\mathcal{O}}(\sqrt{k})$ time per symbol.

Outline of the talk

Introduction

Exact streaming pattern matching

Our streaming k-mismatch algorithm

Conclusions and open problems

Conclusions and open problems

Theorem

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ bits of space and takes $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ time per symbol.

Conclusions and open problems

Theorem

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ *bits of space and takes* $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ *time per symbol.*

Possible directions for further research:

Conclusions and open problems

Theorem

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ bits of space and takes $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ time per symbol.

Possible directions for further research:

1. Fewer logarithmic factors from the query time?

Conclusions and open problems

Theorem

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ *bits of space and takes* $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ *time per symbol.*

Possible directions for further research:

1. Fewer logarithmic factors from the query time?
2. Does small alphabet help in the $\widetilde{\mathcal{O}}(k)$-space regime?
    - Not clear even if random access is allowed.
    - $\widetilde{\mathcal{O}}(1)$ time possible in $\widetilde{\mathcal{O}}(k^2)$ space.

Conclusions and open problems

Theorem

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ bits of space and takes $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ time per symbol.

Possible directions for further research:

1. Fewer logarithmic factors from the query time?
2. Does small alphabet help in the $\widetilde{\mathcal{O}}(k)$-space regime?
   - Not clear even if random access is allowed.
   - $\widetilde{\mathcal{O}}(1)$ time possible in $\widetilde{\mathcal{O}}(k^2)$ space.
3. Improved **amortized** running time?

Conclusions and open problems

Theorem

There is a streaming k-mismatch algorithm which uses $\mathcal{O}(k \log m \log \frac{m}{k})$ bits of space and takes $\mathcal{O}((\sqrt{k \log k} + \log^3 m) \log \frac{m}{k})$ time per symbol.

Possible directions for further research:

1. Fewer logarithmic factors from the query time?
2. Does small alphabet help in the $\widetilde{\mathcal{O}}(k)$-space regime?
    - Not clear even if random access is allowed.
    - $\widetilde{\mathcal{O}}(1)$ time possible in $\widetilde{\mathcal{O}}(k^2)$ space.
3. Improved **amortized** running time?
4. Any $\omega(k + \log n)$ lower bounds?
    - $\Omega(k \log n)$ might be feasible.
    - $\omega(\log n)$ for constant k might give hints for exact matching.

Questions?

Thank you for your attention!

Questions?

Thank you for your attention!

Advertisement

University of Bristol is hiring!

Assistant/associate professorship in algorithms and complexity.
Visit `tinyurl.com/bristolukjob` or talk to Raphaël Clifford.