New algorithms for streaming pattern matching
with many patterns or a few mismatches

Benjamin Sach

joint work with
Raphaël Clifford, Allyx Fontaine,
Ely Porat and Tatiana Starikovskaya.
Exact pattern matching
Exact pattern matching
Exact pattern matching
Exact pattern matching

<table>
<thead>
<tr>
<th>c</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
</table>
Exact pattern matching
Exact pattern matching

a stream of characters, arriving one at a time
Exact pattern matching

A stream of characters, arriving one at a time

\[ c b a a b a \]
Exact pattern matching

a stream of characters, arriving one at a time
Exact pattern matching

a stream of characters, arriving one at a time
Exact pattern matching

a stream of characters, arriving one at a time
Exact pattern matching

a stream of characters, arriving one at a time
Exact pattern matching

a stream of characters, arriving one at a time
Exact pattern matching

A stream of characters, arriving one at a time

The pattern, \( P \)
Exact pattern matching

A stream of characters, arriving one at a time

The pattern, \( P \)

**Goal** find each (exact) occurrence of the pattern, *as it occurs*
Exact pattern matching

A stream of characters, arriving one at a time

Goal find each (exact) occurrence of the pattern, as it occurs
Exact pattern matching

a stream of characters, arriving one at a time

Goal find each (exact) occurrence of the pattern, as it occurs
Exact pattern matching

a stream of characters, arriving one at a time

Goal find each (exact) occurrence of the pattern, as it occurs
Exact pattern matching

a stream of characters, arriving one at a time

Goal find each (exact) occurrence of the pattern, as it occurs
Exact pattern matching

a stream of characters, arriving one at a time

The pattern, $P$

**Goal** find each (exact) occurrence of the pattern, *as it occurs*
Existing algorithms

Goal: find each (exact) occurrence of the pattern, as it occurs

da stream of characters, arriving one at a time

the pattern, $P$
Existing algorithms

A stream of characters, arriving one at a time

Goal: find each (exact) occurrence of the pattern, as it occurs

Pattern: $P = \text{c a b a b b}$
a stream of characters, arriving one at a time

**Existing algorithms**

Goal: find each (exact) occurrence of the pattern, as it occurs

**Deterministic Algorithms**
Existing algorithms

Goal find each (exact) occurrence of the pattern, as it occurs

Deterministic Algorithms

Real-time KMP ([Knuth-Morris-Pratt 1977], [Galil 1981]) takes $O(1)$ time per character and uses $O(|P|)$ space
a stream of characters, arriving one at a time

Existing algorithms

Goal find each (exact) occurrence of the pattern, as it occurs

Deterministic Algorithms

Real-time KMP ([Knuth-Morris-Pratt 1977], [Galil 1981]) takes \(O(1)\) time per character and uses \(O(|P|)\) space

There is an \(\Omega(|P|)\) space lower bound

(you have to store the pattern)
Existing algorithms

a stream of characters, arriving one at a time

Goal find each (exact) occurrence of the pattern, as it occurs

Deterministic Algorithms

Real-time KMP ([Knuth-Morris-Pratt 1977], [Galil 1981]) takes

\[ O(1) \] time per character and uses \[ O(|P|) \] space

There is an \[ \Omega(|P|) \] space lower bound

(you have to store the pattern)

Randomised Algorithms
a stream of characters, arriving one at a time

Existing algorithms

Goal find each (exact) occurrence of the pattern, as it occurs

Deterministic Algorithms

Real-time KMP ([Knuth-Morris-Pratt 1977], [Galil 1981]) takes
\( O(1) \) time per character and uses \( O(|P|) \) space

There is an \( \Omega(|P|) \) space lower bound
(you have to store the pattern)

Randomised Algorithms

[Porat and Porat 2009] developed an algorithm which takes
\( O(\log |P|) \) time per character and uses only \( O(\log |P|) \) space
a stream of characters, arriving one at a time

Existing algorithms

Goal find each (exact) occurrence of the pattern, as it occurs

Deterministic Algorithms

Real-time KMP ([Knuth-Morris-Pratt 1977], [Galil 1981]) takes

\[ O(1) \] time per character and uses \[ O(|P|) \] space

There is an \[ \Omega(|P|) \] space lower bound

\( \text{(you have to store the pattern)} \)

Randomised Algorithms - correct with high probability \( \left( \text{at least } 1 - \frac{1}{n^3} \right) \)

[Porat and Porat 2009] developed an algorithm which takes

\[ O(\log |P|) \] time per character and uses only \[ O(\log |P|) \] space
a stream of characters, arriving one at a time

Existing algorithms

| b | a | b | b | a | a | b | c | b | a | a | b | a | a | a | b | c | a | b | a | b |

Goal find each (exact) occurrence of the pattern, as it occurs

Deterministic Algorithms

Real-time KMP ([Knuth-Morris-Pratt 1977], [Galil 1981]) takes

\[ O(1) \] time per character and uses \( O(|P|) \) space

There is an \( \Omega(|P|) \) space lower bound

\( (\text{you have to store the pattern}) \)

Randomised Algorithms - correct with high probability \((\text{at least} \ 1 - \frac{1}{n^3})\)

[Porat and Porat 2009] developed an algorithm which takes

\[ O(\log |P|) \] time per character and uses only \( O(\log |P|) \) space
a stream of characters, arriving one at a time

Existing algorithms

b a b b a a b c b a a b a a a b c a b a b b

Goal find each (exact) occurrence of the pattern, as it occurs

c a b a b b

the pattern, $P$

Deterministic Algorithms

Real-time KMP ([Knuth-Morris-Pratt 1977], [Galil 1981]) takes $O(1)$ time per character and uses $O(|P|)$ space

There is an $\Omega(|P|)$ space lower bound

(you have to store the pattern)

Randomised Algorithms - correct with high probability (at least $1 - \frac{1}{n^3}$)

[Porat and Porat 2009] developed an algorithm which takes $O(\log |P|)$ time per character and uses only $O(\log |P|)$ space

[Galil and Breslauer 2011] improved the time to $O(1)$ per character
Dictionary matching (many patterns)
Dictionary matching (many patterns)
Dictionary matching (many patterns)
Dictionary matching (many patterns)
Dictionary matching (many patterns)
Dictionary matching (many patterns)

A stream of characters, arriving one at a time
Dictionary matching (many patterns)

a stream of characters, arriving one at a time

```
c b a a b a
```
Dictionary matching (many patterns)

a stream of characters, arriving one at a time
Dictionary matching (many patterns)

a stream of characters, arriving one at a time
Dictionary matching (many patterns)

A stream of characters, arriving one at a time
Dictionary matching (many patterns)

a stream of characters, arriving one at a time

\[\text{c \ b \ a \ a \ b \ a \ a \ a \ b \ c}\]
Dictionary matching (many patterns)

a stream of characters, arriving one at a time
Dictionary matching (many patterns)

A stream of characters, arriving one at a time

A dictionary containing many patterns
Dictionary matching (many patterns)

A stream of characters, arriving one at a time

A dictionary containing many patterns

Goal find each (exact) occurrence of any pattern in the dictionary as it occurs
Dictionary matching (many patterns)

a stream of characters, arriving one at a time

Goal find each (exact) occurrence of any pattern in the dictionary as it occurs
Dictionary matching (many patterns)

a stream of characters, arriving one at a time

Goal find each (exact) occurrence of any pattern in the dictionary as it occurs
Dictionary matching (many patterns)

A stream of characters, arriving one at a time

Goal find each (exact) occurrence of any pattern in the dictionary as it occurs
Dictionary matching (many patterns)

a stream of characters, arriving one at a time

a dictionary containing many patterns

Goal find each (exact) occurrence of any pattern in the dictionary as it occurs
Dictionary matching (many patterns)

A stream of characters, arriving one at a time

Goal: find each (exact) occurrence of any pattern in the dictionary as it occurs

A dictionary containing many patterns
Dictionary matching (many patterns)

a stream of characters, arriving one at a time

a dictionary containing many patterns

Goal find each (exact) occurrence of any pattern in the dictionary as it occurs
Dictionary matching (many patterns)

Goal: find each (exact) occurrence of any pattern in the dictionary as it occurs.
New and existing algorithms

a stream of characters, arriving one at a time

Goal find occurrences of any pattern in the dictionary as they occur

a dictionary containing \( d \) patterns of total length \( M \)
a stream of characters, arriving one at a time

New and existing algorithms

c b a a b a a a b c a b a b b

Goal find occurrences of any pattern in the dictionary as they occur

a dictionary containing \( d \) patterns of total length \( M \)
New and existing algorithms

Goal find occurrences of any pattern in the dictionary as they occur

a dictionary containing $d$ patterns of total length $M$

Deterministic Algorithms
New and existing algorithms

Deterministic Algorithms

[Aho and Corasick 1975] takes $O(1)$ time per character and uses $O(M)$ space.
New and existing algorithms

A stream of characters, arriving one at a time

c b a a b a a a b c a b a b b

goal find occurrences of any pattern in the dictionary as they occur

[\text{Deterministic Algorithms}]

[Aho and Corasick 1975] takes $O(1)$ time per character and uses $O(M)$ space
a stream of characters, arriving one at a time

New and existing algorithms

| c | b | a | a | b | a | a | a | b | c | a | b | a | b | b |

Goal find occurrences of any pattern in the dictionary as they occur

| c | a | b | a | b | b |
| b | c | a | b |
| a | a | b | b | a | a | b | c |

a dictionary containing \( d \) patterns of total length \( M \)

Deterministic Algorithms

[Aho and Corasick 1975] takes \( O(1) \) time per character and uses \( O(M) \) space
New and existing algorithms

Goal find occurrences of any pattern in the dictionary as they occur

a dictionary containing $d$ patterns of total length $M$

Deterministic Algorithms

[Aho and Corasick 1975] takes $O(1)$ time per character and uses $O(M)$ space

There is an $\Omega(|M|)$ space lower bound
(you have to store all the patterns)
New and existing algorithms

a stream of characters, arriving one at a time

Goal find occurrences of any pattern in the dictionary as they occur

a dictionary containing \( d \) patterns of total length \( M \)

Deterministic Algorithms

[Aho and Corasick 1975] takes \( O(1) \) time per character and uses \( O(M) \) space

There is an \( \Omega(|M|) \) space lower bound (you have to store all the patterns)

Randomised Algorithms
New and existing algorithms

| c | b | a | a | b | a | a | b | c | a | b | a | b | b |

Goal find occurrences of any pattern in the dictionary as they occur

a dictionary containing \(d\) patterns of total length \(M\)

Deterministic Algorithms

[Aho and Corasick 1975] takes \(O(1)\) time per character and uses \(O(M)\) space.

There is an \(\Omega(|M|)\) space lower bound (you have to store all the patterns).

Randomised Algorithms

New and existing algorithms

A stream of characters, arriving one at a time

Goal: find occurrences of any pattern in the dictionary as they occur

A dictionary containing \(d\) patterns of total length \(M\)

Deterministic Algorithms

[Aho and Corasick 1975] takes \(O(1)\) time per character and uses \(O(M)\) space

There is an \(\Omega(\lvert M \rvert)\) space lower bound
(you have to store all the patterns)

Randomised Algorithms

Run [Galil and Breslauer 2011] algorithm once for each pattern in parallel
this uses only \(O(d \log m)\) space
a stream of characters, arriving one at a time

New and existing algorithms

| c | b | a | a | b | a | a | a | b | c | a | b | a | a | b | b |

Goal find occurrences of any pattern in the dictionary as they occur

| c | a | b | a | b | b |
| b | c | a | b |
| a | a | b | b | a | a | b |

a dictionary containing \( d \) patterns of total length \( M \) and maximum length \( m \)

Deterministic Algorithms

[Aho and Corasick 1975] takes \( O(1) \) time per character and uses \( O(M) \) space

There is an \( \Omega(|M|) \) space lower bound (you have to store all the patterns)

Randomised Algorithms

Run [Galil and Breslauer 2011] algorithm once for each pattern in parallel

this uses only \( O(d \log m) \) space
a stream of characters, arriving one at a time

New and existing algorithms

| c | b | a | a | b | a | a | b | c | a | b | a | b | b |

Goal find occurrences of any pattern in the dictionary as they occur

| c | a | b | a | b | b |
| b | c | a | b |
| a | a | b | b | a | a | b | c |

a dictionary containing \(d\) patterns of total length \(M\) and maximum length \(m\)

Deterministic Algorithms

[Aho and Corasick 1975] takes \(O(1)\) time per character and uses \(O(M)\) space.

There is an \(\Omega(|M|)\) space lower bound:
(\textbf{you have to store all the patterns})

Randomised Algorithms


this uses only \(O(d \log m)\) space
New and existing algorithms

a stream of characters, arriving one at a time

Goal find occurrences of any pattern in the dictionary as they occur

a dictionary containing \( d \) patterns of total length \( M \) and maximum length \( m \)

Deterministic Algorithms

[Aho and Corasick 1975] takes \( O(1) \) time per character and uses \( O(M) \) space.

There is an \( \Omega(|M|) \) space lower bound (you have to store all the patterns)

Randomised Algorithms

Run [Galil and Breslauer 2011] algorithm once for each pattern in parallel
this uses only \( O(d \log m) \) space
New and existing algorithms

A stream of characters, arriving one at a time

Goal find occurrences of any pattern in the dictionary as they occur

A dictionary containing \( d \) patterns of total length \( M \) and maximum length \( m \)

Deterministic Algorithms

[Aho and Corasick 1975] takes \( O(1) \) time per character and uses \( O(M) \) space

There is an \( \Omega(|M|) \) space lower bound (you have to store all the patterns)

Randomised Algorithms

Run [Galil and Breslauer 2011] algorithm once for each pattern in parallel

this uses only \( O(d \log m) \) space

There is a randomised \( \Omega(d) \) bit space lower bound (you have to store something about each pattern)
New and existing algorithms

a stream of characters, arriving one at a time

| c | b | a | a | b | a | a | b | c | a | b | a | b | b |

Goal find occurrences of any pattern in the dictionary as they occur

| c | a | b | a | b | b |
| b | c | a | b |
| a | a | b | b | a | a | b | c |

a dictionary containing \(d\) patterns of total length \(M\) and maximum length \(m\)

Deterministic Algorithms

[Aho and Corasick 1975] takes \(O(1)\) time per character and uses \(O(M)\) space

There is an \(\Omega(|M|)\) space lower bound (you have to store all the patterns)

Randomised Algorithms

Run [Galil and Breslauer 2011] algorithm once for each pattern in parallel

this uses only \(O(d \log m)\) space
New and existing algorithms

Goal find occurrences of any pattern in the dictionary as they occur

a dictionary containing \(d\) patterns of total length \(M\) and maximum length \(m\)

Deterministic Algorithms

[Aho and Corasick 1975] takes \(O(1)\) time per character and uses \(O(M)\) space

There is an \(\Omega(|M|)\) space lower bound (you have to store all the patterns)

Randomised Algorithms

Run [Galil and Breslauer 2011] algorithm once for each pattern in parallel

this uses only \(O(d \log m)\) space but takes \(O(d)\) time per character
New and existing algorithms

A stream of characters, arriving one at a time

Goal: find occurrences of any pattern in the dictionary as they occur

A dictionary containing $d$ patterns of total length $M$ and maximum length $m$

Deterministic Algorithms

[Aho and Corasick 1975] takes $O(1)$ time per character and uses $O(M)$ space

There is an $\Omega(|M|)$ space lower bound (you have to store all the patterns)

Randomised Algorithms

Run [Galil and Breslauer 2011] algorithm once for each pattern in parallel

this uses only $O(d \log m)$ space but takes $O(d)$ time per character

(NEW!) Our dictionary matching algorithm also uses $O(d \log m)$ space and takes only $O(\log \log(d + m))$ time per character
Streaming pattern matching with few mistakes ($k$-mismatch)
Streaming pattern matching with few mistakes ($k$-mismatch)
Streaming pattern matching with few mistakes ($k$-mismatch)
Streaming pattern matching with few mistakes \((k\text{-mismatch})\)
Streaming pattern matching with few mistakes ($k$-mismatch)
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time:

```
c b a a b a a a
```
Streaming pattern matching with few mistakes \( (k\text{-mismatch}) \)

a stream of characters, arriving one at a time
Streaming pattern matching with few mistakes \((k\text{-mismatch})\)

a stream of characters, arriving one at a time

\begin{itemize}
  \item \texttt{c b a a b a a a b c}
\end{itemize}
Streaming pattern matching with few mistakes \((k\text{-mismatch})\)

a stream of characters, arriving one at a time

c b a a b a a a b c a
Streaming pattern matching with few mistakes ($k$-mismatch)

a stream of characters, arriving one at a time
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time

The pattern, $P$

```
c b a a b a a a b c a b
```

```
c a b a b b
```
Streaming pattern matching with few mistakes ($k$-mismatch)

a stream of characters, arriving one at a time

the pattern, $P$
(there is only one pattern)
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time

```
c b a a b a a a b c a b
```

The pattern, $P$

```
c a b a b b
```
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time

The pattern, $P$

**Goal** Output the *latest* Hamming distance
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time

The pattern, $P$

**Goal** Output the *latest* Hamming distance

Number of mismatches
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time:

```
c b a a b a a a b c a b
```

The pattern, $P$:

```
c a b a b b
```

Goal: Output the latest Hamming distance number of mismatches

3 mismatches
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time

The pattern, $P$

3 mismatches

Goal: Output the latest Hamming distance

Number of mismatches
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time:

```
c b a a b a a a b c a b a a
```

The pattern, $P$

```
c a b a b b
```

Goal: Output the latest Hamming distance number of mismatches

4 mismatches
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time

A stream of characters: $cbabaabcaabb$

The pattern: $cbabaab$

Number of mismatches: $k$ mismatches

Goal: Output the latest Hamming distance
Streaming pattern matching with few mistakes ($k$-mismatch)

Goal: Output the latest Hamming distance, if it is at most $k$.

A stream of characters, arriving one at a time:

```
c b a a b a a b c a b a b
```

The pattern, $P$

```
c a b a b b
```

$mismatches > k$
Streaming pattern matching with few mistakes ($k$-mismatch)

A stream of characters, arriving one at a time

The pattern, $P$

Goal: Output the latest Hamming distance, if it is at most $k$, number of mismatches

A threshold given in the input
Streaming pattern matching with few mistakes (\(k\)-mismatch)

a stream of characters, arriving one at a time

the pattern, \(P\)

0 mismatches (an exact match)

Goal Output the latest Hamming distance, number of mismatches, if it is at most \(k\)

a threshold given in the input
New and existing algorithms

3 mismatches

Goal: Output the latest Hamming distance, if it is at most $k$.}

P: $\begin{bmatrix} c & a & b & a & b & b \end{bmatrix}$
New and existing algorithms

3 mismatches

Goal: Output the latest Hamming distance, if it is at most k.
New and existing algorithms

3 mismatches

Goal: Output the latest Hamming distance, if it is at most $k$.

Offline Algorithms (i.e. the input is given all at once)
New and existing algorithms

3 mismatches

Goal Output the latest Hamming distance, if it is at most $k$

Offline Algorithms (i.e. the input is given all at once)

The fastest known offline algorithm for $k$-mismatch ([Amir, Levenstein and Porat 2000]) takes $O(n\sqrt{k \log k})$ total time on an input text of length $n$
New and existing algorithms

Goal Output the latest Hamming distance, if it is at most $k$

### Offline Algorithms (i.e. the input is given all at once)

The fastest known offline algorithm for $k$-mismatch ([Amir, Levenstein and Porat 2000])

takes $O(n\sqrt{k \log k})$ total time on an input text of length $n$

(and it is deterministic)
New and existing algorithms

3 mismatches

Goal: Output the latest Hamming distance, if it is at most $k$.

Offline Algorithms (i.e., the input is given all at once)

The fastest known offline algorithm for $k$-mismatch ([Amir, Levenstein and Porat 2000])

takes $O(n\sqrt{k \log k})$ total time on an input text of length $n$

(and it is deterministic)

Randomised Streaming Algorithms
New and existing algorithms

Goal: Output the latest Hamming distance, if it is at most $k$

Offline Algorithms (i.e. the input is given all at once)

The fastest known offline algorithm for $k$-mismatch ([Amir, Levenstein and Porat 2000])

takes $O(n\sqrt{k \log k})$ total time on an input text of length $n$

(and it is deterministic)

Randomised Streaming Algorithms

[Porat and Porat 2009] also gave a $k$-mismatch algorithm which uses $\tilde{O}(k^3)$ space

and takes $\tilde{O}(k^2)$ time per character
New and existing algorithms

Goal: Output the latest Hamming distance, if it is at most $k$.

Offline Algorithms (i.e. the input is given all at once)

The fastest known offline algorithm for $k$-mismatch ([Amir, Levenstein and Porat 2000]) takes $O(n\sqrt{k \log k})$ total time on an input text of length $n$ (and it is deterministic).

Randomised Streaming Algorithms

[Porat and Porat 2009] also gave a $k$-mismatch algorithm which uses $\tilde{O}(k^3)$ space and takes $\tilde{O}(k^2)$ time per character.

$\tilde{O}(\mathbb{N})$ hides logarithmic factors.
New and existing algorithms

Goal: Output the *latest* Hamming distance, *if it is at most* \( k \)

### Offline Algorithms
(i.e. the input is given all at once)

The fastest known offline algorithm for \( k \)-mismatch ([Amir, Levenstein and Porat 2000]) takes \( O(n\sqrt{k \log k}) \) total time on an input text of length \( n \) (and it is deterministic)

### Randomised Streaming Algorithms

[Porat and Porat 2009] also gave a \( k \)-mismatch algorithm which uses \( \tilde{O}(k^3) \) space and takes \( \tilde{O}(k^2) \) time per character
New and existing algorithms

Goal: Output the latest Hamming distance, if it is at most \( k \).

**Offline Algorithms** (i.e. the input is given all at once)

The fastest known offline algorithm for \( k \)-mismatch ([Amir, Levenstein and Porat 2000]) takes \( O(n\sqrt{k \log k}) \) total time on an input text of length \( n \) (and it is deterministic).

**Randomised Streaming Algorithms**

[Porat and Porat 2009] also gave a \( k \)-mismatch algorithm which uses \( \tilde{O}(k^3) \) space and takes \( \tilde{O}(k^2) \) time per character.

(NEW!) Our \( k \)-mismatch algorithm uses \( \tilde{O}(k^2) \) space and takes only \( \tilde{O}(\sqrt{k}) \) time per character.
New and existing algorithms

Goal: Output the latest Hamming distance, if it is at most $k$.

Offline Algorithms (i.e. the input is given all at once)

The fastest known offline algorithm for $k$-mismatch ([Amir, Levenstein and Porat 2000])
takes $O(n \sqrt{k \log k})$ total time on an input text of length $n$
(and it is deterministic)

Randomised Streaming Algorithms

[Porat and Porat 2009] also gave a $k$-mismatch algorithm which uses $\tilde{O}(k^3)$ space
and takes $\tilde{O}(k^2)$ time per character

(NEW!) Our $k$-mismatch algorithm uses $\tilde{O}(k^2)$ space
and takes only $\tilde{O}(\sqrt{k})$ time per character

There is a randomised $\Omega(k)$ bit space lower bound.
Two types of patterns

Either,

the pattern, \( P \)

is very \( \text{compressible} \)

any two alignments

with at most \( k \) mismatches

are at least \( k \) characters apart

\( \leq k \) mismatches

\( \geq k \)
Case 1: Compressible patterns

don’t store the pattern, $P$

store this instead $O(k)$
Case 1: Compressible patterns

don’t we also need to store all this?

don’t store the pattern, $P$

store this instead $O(k)$
**Case 1: Compressible patterns**

don’t we also need to store all this?

yes, but either

don’t store the pattern, $P$

store this instead $O(k)$
Case 1: Compressible patterns

don’t we also need to store all this?

yes, but either

it’s very compressible

O(k)

store this instead

O(k)
Case 1: Compressible patterns

don’t we also need to store all this?

yes, but either

it’s very compressible or

O(k)

\[
\text{don’t store the pattern, } P
\]

\[
\text{store this instead } O(k)
\]
Case 1: Compressible patterns

Don't we also need to store all this? Yes, but either it's very compressible or it isn't very similar to $P$ ($> k$ mismatches).
Two types of patterns

Either,

the pattern, $P$

is very compressible

any two alignments with at most $k$ mismatches are at least $k$ characters apart

$\leq k$ mismatches

$\geq k$
Case 2: Rare patterns
Case 2: Rare patterns

\[ \leq k \text{ mismatches} \]

\[ \leq k \text{ mismatches} \]
Case 2: Rare patterns

\[ \leq k \text{ mismatches} \]

\[ \leq k \text{ mismatches} \]
Case 2: Rare patterns

\[ \leq k \text{ mismatches} \]

\[ \leq 2k \text{ mismatches} \]
Case 2: Rare patterns

\[ \leq k \text{ mismatches} \]

\[ \leq 2k \text{ mismatches} \]

\[ \geq k \]

\[ \leq k \text{ mismatches} \]

\[ \leq 2k \text{ mismatches} \]
Case 2: Rare patterns

Step 1 We find every \( \times \) and some of the \( \odot \)

by approximating the latest Hamming distance
Case 2: Rare patterns

Step 1 We find every $\times$ and some of the $\bullet$ by approximating the latest Hamming distance.

Step 2 Whenever we find a $\times$/$\bullet$ we calculate the true Hamming distance by sampling mismatches.
**Case 2: Rare patterns**

**Step 1** We find every $\times$ and some of the $\bullet$ by approximating the latest Hamming distance.

**Step 2** Whenever we find a $\times/\bullet$ we calculate the true Hamming distance by sampling mismatches.

Step 2 is slow ($\tilde{O}(k)$ time) but happens rarely.
How do we approximate the Hamming distance?

The pattern, $P$
How do we approximate the Hamming distance?

pick a random prime \((\text{around } k \log^2 |\mathcal{P}|)\)
How do we approximate the Hamming distance?

the pattern, $P$

pick a random prime (around $k \log^2 |P|$)
How do we approximate the Hamming distance?

The pattern, $P$

Pick a random prime (around $k \log^2 |P|$)

Characters which are the same colour are in the same sub-pattern
How do we approximate the Hamming distance?

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match

pick a random prime \((\text{around } k \log^2 |P|)\)
How do we approximate the Hamming distance?

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match

pick a random prime 
(around \( k \log^2 |P| \))
How do we approximate the Hamming distance?

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match

pick a random prime \( (\text{around } k \log^2 |P|) \)
How do we approximate the Hamming distance?

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match
How do we approximate the Hamming distance?

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match

The pattern, $P$

Pick a random prime (around $k \log^2 |P|$)
How do we approximate the Hamming distance?

The pattern, $P$

pick a random prime (around $k \log^2 |P|$)

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match
How do we approximate the Hamming distance?

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match
How do we approximate the Hamming distance?

2 sub-patterns do not exactly match ( and )

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match

pick a random prime (around $k \log^2 |P|$)
How do we approximate the Hamming distance?

2 sub-patterns do not exactly match ( and )

this undercounts the true number of mismatches (which is 3)

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match

pick a random prime (around $k \log^2 |P|$)
How do we approximate the Hamming distance?

2 sub-patterns do not exactly match ( and )

this undercounts the true number of mismatches (which is 3)

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match
How do we approximate the Hamming distance?

2 sub-patterns do not exactly match ( and )

This undercounts the true number of mismatches (which is 3)

Characters which are the same colour are in the same sub-pattern

Count the number of sub-patterns that do not exactly match

(we actually repeat this a logarithmic number of times and take the maximum)

pick a random prime 

(around \( k \log^2 |P| \))
How do we approximate the Hamming distance?

2 sub-patterns do not *exactly* match ( and )

*This undercounts the true number of mismatches (which is 3)*

Count the number of sub-patterns that do not *exactly* match

(we actually repeat this a logarithmic number of times and take the maximum)

How do we determine which sub-patterns match?

*Characters which are the same colour are in the same sub-pattern*

**pick a random prime**

(around $k \log^2 |P|$)
How do we determine which sub-patterns match?
How do we determine which sub-patterns match?
How do we determine which sub-patterns match?
How do we determine which sub-patterns match?
How do we determine which sub-patterns match?

\[
\begin{align*}
\text{c a b a b b} \\
\text{b a a c b c c b a c c b a c a a b a c b c}
\end{align*}
\]
How do we determine which sub-patterns match?
How do we determine which sub-patterns match?

\[
\begin{array}{cccc}
\text{c} & \text{a} & \text{b} & \text{a} & \text{b} & \text{b} \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{b} & \text{a} & \text{a} & \\
\text{b} & \text{c} & \text{c} & \\
\text{a} & \text{c} & \text{c} & \\
\text{a} & \text{c} & \text{a} & \\
\text{b} & \text{a} & \text{c} & \\
\text{c} & \\
\text{c} & \text{b} & \text{b} & \text{a} & \text{b} & \\
\end{array}
\]
How do we determine which sub-patterns match?
How do we determine which sub-patterns match?
How do we determine which sub-patterns match?

\[ \begin{array}{cccccc}
  c & a & b & a & b & b \\
  b & b & a & a & b & c \\
  a & a & c & c & c & a & a & c \\
  c & b & b & a & b \\
\end{array} \]
How do we determine which sub-patterns match?
How do we determine which sub-patterns match?
How do we determine which sub-patterns match?

```
c a b a b b

b b a a b c
```

```
a c c a c
```

```
c b b a b
```
How do we determine which sub-patterns match?

- $c\ a\ b\ a\ b\ b$
- $b\ b\ a\ a\ b\ c$
- $a\ c\ c\ c\ a$
- $a\ c\ c\ a\ c$
- $c\ b\ b\ a\ b$
How do we determine which sub-patterns match?

\[
\begin{array}{cccc}
    c & a & b & a & b & b \\
    b & b & a & a & b & c \\
    a & c & c & c & a \\
    a & c & c & a & c \\
    c & b & b & a & b \\
\end{array}
\]
How do we determine which sub-patterns match?

A dictionary containing \( \tilde{O}(k) \) patterns.
How do we determine which sub-patterns match?

We partition the stream analogously, and find all $\tilde{O}(k)$ sub-patterns in all $\tilde{O}(k)$ sub-streams.
How do we determine which sub-patterns match?

We partition the stream analogously, and find all $\tilde{O}(k)$ sub-patterns in all $\tilde{O}(k)$ sub-streams using our dictionary matching algorithm.
How do we determine which sub-patterns match?

We partition the stream analogously, and find all $\tilde{O}(k)$ sub-patterns in all $\tilde{O}(k)$ sub-streams using our dictionary matching algorithm.

(This is where the $\tilde{O}(k^2)$ space comes from)
Conclusions

We developed a new algorithm for finding many patterns (dictionary matching) in a stream

The algorithm uses $O(d \log m)$ space

and takes only $O(\log \log(d + m))$ time per character

*where* $d$ *is the number of patterns and* $m$ *is the longest pattern length*

We developed new algorithms for pattern matching with few mistakes ($k$-mismatch) in a stream

The algorithm discussed uses $\tilde{O}(k^2)$ space and takes $\tilde{O}(\sqrt{k})$ time per character

Our second algorithm also uses $\tilde{O}(k^2)$ space and takes $\tilde{O}(k^2/|P|)$ amortised time per character

We also give a $(1 + \epsilon)$ approximation algorithm which uses $\tilde{O}(k^2/\epsilon^2)$ space

and takes $\tilde{O}(1/\epsilon^2)$ time per character

*where* $k$ *is maximum number of mismatches tolerated*