Pattern Matching in Streaming Data

Raphaël Clifford and Benjamin Sach

{clifford,sach}@cs.bris.ac.uk

Department of Computer Science, University of Bristol
Searching in a stream
Introduction: Online pattern matching

- Consider a text, $T$ and a pattern $P$ (length $m$)
- We assume we have $P$ in advance but $T$ arrives online...

$T : a\ b\ c\ ??\ ??\ ??\ ??\ ??\ ??\ ??$

$P : a\ b\ a\ $ (dist = 1)

- Find the **distance**, $d(i)$, between $T[i, i + m - 1]$ and $P$ for all $i$.
  (Hamming distance shown)

- We are concerned with **worst-case** time per text character.
  (Pseudo-Realtime)
Consider a text, \( T \) and a pattern \( P \) (length \( m \))

We assume we have \( P \) in advance but \( T \) arrives online...

\[
T: \quad \text{a b c a } \, ? \, ? \, ? \, ? \, ? \, ? \, ? \\
P: \quad \text{a b a } \\
\text{(dist = 2)}
\]

Find the distance, \( d(i) \), between \( T[i, i + m - 1] \) and \( P \) for all \( i \).

(Hamming distance shown)

We are concerned with worst-case time per text character.

(Pseudo-Realtime)
Consider a text, $T$ and a pattern $P$ (length $m$).

We assume we have $P$ in advance but $T$ arrives online...

$$T : \quad a \ b \ c \ a \ a \ ? \ ? \ ? \ ? \ ? \ ? \ ? \ ?$$

$$P : \quad a \ b \ a \quad (\text{dist} = 2)$$

Find the distance, $d(i)$, between $T[i, i + m - 1]$ and $P$ for all $i$.

(Hamming distance shown)

We are concerned with worst-case time per text character.

(Pseudo-Realtime)
Consider a text, $T$ and a pattern $P$ (length $m$)

We assume we have $P$ in advance but $T$ arrives online...

$T : \begin{array}{cccccccccc}
\text{a} & \text{b} & \text{c} & \text{a} & \text{a} & \text{b} & ? & ? & ? & ?
\end{array}$

$P : \begin{array}{c}
\text{a} & \text{b} & \text{a} \\
\end{array}$ (dist = 2)

Find the distance, $d(i)$, between $T[i, i + m - 1]$ and $P$ for all $i$.

(Hamming distance shown)

We are concerned with worst-case time per text character.

(Pseudo-Realtime)
Consider a text, $T$ and a pattern $P$ (length $m$)

We assume we have $P$ in advance but $T$ arrives online...

$T : \text{a b c a a b a ? ? ? ?}$

$P : \text{a b a}$ (dist = 0)

Find the distance, $d(i)$, between $T[i, i + m - 1]$ and $P$ for all $i$.

(Hamming distance shown)

We are concerned with worst-case time per text character.

(Pseudo-Realtime)
Consider a text, $T$ and a pattern $P$ (length $m$)

We assume we have $P$ in advance but $T$ arrives online...

$T: \text{a b c a a b a b ? ?}$

$P: \text{a b a}$ (dist = 3)

Find the distance, $d(i)$, between $T[i, i + m - 1]$ and $P$ for all $i$.

(Hamming distance shown)

We are concerned with worst-case time per text character.

(Pseudo-Realtime)
Consider a text, $T$ and a pattern $P$ (length $m$)

We assume we have $P$ in advance but $T$ arrives online...

$T : \text{a b c a a b a b a a ?}$

$P : \text{a b a}$

Find the distance, $d(i)$, between $T[i, i + m - 1]$ and $P$ for all $i$.

(Hamming distance shown)

We are concerned with worst-case time per text character.

(Pseudo-Realtime)
Consider a text, $T$ and a pattern $P$ (length $m$).

We assume we have $P$ in advance but $T$ arrives online...

Find the distance, $d(i)$, between $T[i, i + m - 1]$ and $P$ for all $i$.

We are concerned with worst-case time per text character.
Local and non-local pattern matching

A distance is **local** if (roughly) it can be written as:

\[
d(i) = \sum_{j=0}^{m-1} \Delta(P[j], T[i+j])
\]

\(\Delta\) is some function acting on alphabet symbols

- Hamming, \(L_1\), \(L_2\), less-than and k-mismatch are all local.
- Edit distance and k-differences are non-local.
Local and non-local pattern matching

A distance is **local** if (roughly) it can be written as:

\[
d(i) = \sum_{j=0}^{m-1} \Delta(P[j], T[i+j])
\]

\(\Delta\) is some function acting on alphabet symbols)

- Hamming, \(L_1\), \(L_2\), less-than and k-mismatch are all local.
- Edit distance and k-differences are non-local.
Local and non-local pattern matching

A distance is **local** if (roughly) it can be written as:

\[
d(i) = \sum_{j=0}^{m-1} \Delta(P[j], T[i+j])
\]

\(\Delta\) is some function acting on alphabet symbols

- Hamming, \(L_1\), \(L_2\), less-than and k-mismatch are all local.
- Edit distance and k-differences are non-local.
Local online pattern matching

- Split the pattern into $O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.

- Compute distances by summing the distance from each $S_j$ to $T$.

---

1C., Efremenko, Porat and Porat. CPM 2008
Local online pattern matching

**Example:** (using Hamming distance)

| T: b a c b b a b a a c a c a c a b a a b c b c |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P: a b b a b c a b | (dist = 4) |

2 + 1 + 0 + 1

**Plan:**
- Compute distances from each $S_j$ to $T$ using an offline algorithm as a black box.
- Split $T$ into overlapping partitions so that each distance is computed before it is needed.
Local online pattern matching

Example: (using Hamming distance)

T: b a c b b a b a a c a c a c b a a b c b c

P: a b b a b c a b (dist = 4)

2 + 1 + 0 + 1

Plan:

- Compute distances from each $S_j$ to $T$ using an offline algorithm as a black box.
- Split $T$ into overlapping partitions so that each distance is computed before it is needed.
Partition text into overlapping substrings for each $S_j$.

- Use offline algorithm as black box in each text partition.
- Distribute the work across the next $|S_j|/2$ characters.
- Time complexity increases by at worst multiplicative $O(\log m)$.
Local online pattern matching

Partition text into overlapping substrings for each $S_j$.

Use offline algorithm as black box in each text partition.

Distribute the work across the next $|S_j|/2$ characters.

Time complexity increases by at worst multiplicative $O(\log m)$. 
Local online pattern matching

- Partition text into overlapping substrings for each $S_j$.
- Use offline algorithm as black box in each text partition.
- Distribute the work across the next $|S_j|/2$ characters.
- Time complexity increases by at worst multiplicative $O(\log m)$.
Local online pattern matching

 Partition text into overlapping substrings for each $S_j$.

 Use offline algorithm as black box in each text partition.

 Distribute the work across the next $|S_j|/2$ characters.

 Time complexity increases by at worst multiplicative $O(\log m)$.
Local online pattern matching

Partition text into overlapping substrings for each $S_j$.

Use offline algorithm as black box in each text partition.

Distribute the work across the next $|S_j|/2$ characters.

Time complexity increases by at worst multiplicative $O(\log m)$. 

Raphaël Clifford and Benjamin Sach
Pattern Matching in Streaming Data
Partition text into overlapping substrings for each $S_j$.
Use offline algorithm as black box in each text partition.
Distribute the work across the next $|S_j|/2$ characters.
Time complexity increases by at worst multiplicative $O(\log m)$. 
Local online pattern matching

Partition text into overlapping substrings for each \( S_j \).
Use offline algorithm as black box in each text partition.
Distribute the work across the next \(|S_j|/2\) characters.
Time complexity increases by at worst multiplicative \( O(\log m) \).

Raphaël Clifford and Benjamin Sach
Pattern Matching in Streaming Data
Local Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Offline per char time</th>
<th>Online penalty</th>
<th>Online space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact matching with wildcards</td>
<td>(O(\log m))</td>
<td>(O(\log m))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>Hamming distance</td>
<td>(O(\sqrt{m \log m}))</td>
<td>(O(1))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>(k)-mismatch</td>
<td>(O(\sqrt{k \log k}))</td>
<td>(O(\log m))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>(L_1) distance</td>
<td>(O(\sqrt{m \log m}))</td>
<td>(O(1))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>(L_2) distance</td>
<td>(O(\log m))</td>
<td>(O(\log m))</td>
<td>(O(m))</td>
</tr>
</tbody>
</table>

Are there (communication complexity) lower bounds? Do we need this log factor multiplicative overhead?

What about non-local problems?

Can randomisation improve the space complexity?
### Local Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Offline per char time</th>
<th>Online penalty</th>
<th>Online space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact matching with wildcards</td>
<td>$O(\log m)$</td>
<td>$O(\log m)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>Hamming distance</td>
<td>$O(\sqrt{m \log m})$</td>
<td>$O(1)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>$k$-mismatch</td>
<td>$O(\sqrt{k \log k})$</td>
<td>$O(\log m)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>$L_1$ distance</td>
<td>$O(\sqrt{m \log m})$</td>
<td>$O(1)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>$L_2$ distance</td>
<td>$O(\log m)$</td>
<td>$O(\log m)$</td>
<td>$O(m)$</td>
</tr>
</tbody>
</table>

Are there (communication complexity) lower bounds? Do we need this log factor multiplicative overhead?

What about non-local problems?

Can randomisation improve the space complexity?
<table>
<thead>
<tr>
<th>Problem</th>
<th>Offline per char time</th>
<th>Online penalty</th>
<th>Online space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact matching with wildcards</td>
<td>(O(\log m))</td>
<td>(O(\log m))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>Hamming distance</td>
<td>(O(\sqrt{m \log m}))</td>
<td>(O(1))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>(k)-mismatch</td>
<td>(O(\sqrt{k \log k}))</td>
<td>(O(\log m))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>(L_1) distance</td>
<td>(O(\sqrt{m \log m}))</td>
<td>(O(1))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>(L_2) distance</td>
<td>(O(\log m))</td>
<td>(O(\log m))</td>
<td>(O(m))</td>
</tr>
</tbody>
</table>

Are there (communication complexity) lower bounds? Do we need this log factor multiplicative overhead?

What about non-local problems?

Can randomisation improve the space complexity?
## Local Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>Offline per char time</th>
<th>Online penalty</th>
<th>Online space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact matching with wildcards</td>
<td>(O(\log m))</td>
<td>(O(\log m))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>Hamming distance</td>
<td>(O(\sqrt{m \log m}))</td>
<td>(O(1))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>(k)-mismatch</td>
<td>(O(\sqrt{k \log k}))</td>
<td>(O(\log m))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>(L_1) distance</td>
<td>(O(\sqrt{m \log m}))</td>
<td>(O(1))</td>
<td>(O(m))</td>
</tr>
<tr>
<td>(L_2) distance</td>
<td>(O(\log m))</td>
<td>(O(\log m))</td>
<td>(O(m))</td>
</tr>
</tbody>
</table>

Are there (communication complexity) **lower bounds**? Do we need this log factor multiplicative overhead?

What about **non-local problems**?

Can randomisation improve the **space complexity**?
Can we improve the $k$-mismatch algorithm?

- The $k$-mismatch problem is to find all alignments of $P$ with $T$ where the **Hamming distance** is at most $k$.

  $$
  T : \ a \ b \ c \ a \ a \ b \ a \ b \ a \ c \\
  P : \ c \ a \ b \ b \ a \ a \ \\
  \text{(dist} = 2)$$

- Offline: $O(n\sqrt{k \log k})$ time$^2$.
- Pseudo-realtime: $O(\sqrt{k \log k \log m})$ time per character.

**Problem:** $k$ is often very small in comparison with $m$.

$^2$Amir, Lewenstein, Porat. SODA 2000
Can we improve the $k$-mismatch algorithm?

- Split the pattern into $s \in O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.
- However, we set $s$ so that $k^5/2 \leq |S_s| < k^5$.

\[ P: S_1 \quad S_2 \quad \cdots \quad S_s \]

- Use the $O(\sqrt{k \log k \log m})$ algorithm for the final subpattern.
- As $|S_s| \in \Theta(k^5)$ we have $O(\sqrt{k \log k \log k})$ time per character.
Can we improve the $k$-mismatch algorithm?

- Split the pattern into $s \in O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.
- However, we set $s$ so that $k^5/2 \leq |S_s| < k^5$.

Use the $O(\sqrt{k \log k \log m})$ algorithm for the final subpattern.
- As $|S_s| \in \Theta(k^5)$ we have $O(\sqrt{k \log k \log k})$ time per character.
Can we improve the $k$-mismatch algorithm?

- Split the pattern into $s \in O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.
- However, we set $s$ so that $k^5/2 \leq |S_s| < k^5$.

\[ P: \quad S_1 \quad S_2 \quad \cdots \quad S_s \]

\[ \frac{m}{2} \quad \frac{m}{4} \quad \Theta(k^5) \]

- Use the $O(\sqrt{k \log k \log m})$ algorithm for the final subpattern.
- As $|S_s| \in \Theta(k^5)$ we have $O(\sqrt{k \log k \log k})$ time per character.
Can we improve the $k$-mismatch algorithm?

- What about mismatches with subpatterns $S_1, S_2, \ldots, S_{s-1}$?
- Each of these subpatterns has length greater than $k^5$.

![Diagram showing pattern P with subpatterns $S_1, S_2, \ldots, S_s$.]

- For the case where $m \gg k$ there is an offline algorithm\(^3\) with time complexity $O(n + nk^4 \log k/m)$.
- Applied to one of our subpatterns (offline) this would be $O(n)$. Using the black box method we obtain $O(1)$ time per character.

---

\(^3\)Amir, Lewenstein and Porat. SODA 2000
Can we improve the $k$-mismatch algorithm?

- Summing across all subpatterns we achieve a time complexity of $O(\sqrt{k} \log^{3/2} k + \log m)$ per character.

- In very recent work we have also developed a $O(\sqrt{k} \log m)$ time per character $k$-mismatch algorithm using real-time LCE processing and new filtering techniques.

- Using this algorithm coupled with the above technique the time complexity can be further improved to

\[ O(\sqrt{k} \log k + \log m) \]

**Open Problem:** Remove the additive $O(\log m)$
What about **non-local** problems?

**Example Problem: (Swap-Mismatch)**

For each $i$, find the minimum number of moves to transform $P$ into $T[i, i + m - 1]$. No two moves can be applied to the same character. The valid moves are:

- *swap* (exchange two adjacent characters)
- *mismatch* (replace a character).

*T*: $x a n b a d b a b a b c e a p w$

$P$: $a b c b b a c b$

**Offline**: $O(n\sqrt{m\log m})$ time$^4$.

$^4$Amir, Eisenberg and Porat, Algorithmica 2006
What about non-local problems?

- Consider splitting the pattern into $O(\log m)$ subpatterns...
- What about the swaps at the boundaries?
  - Only four possible cases

1. Compute distances for all transformed subpatterns using the black box method.
2. Stitch the solutions for the subpatterns together by calculating the optimal swaps at each boundary.

**Pseudo-realtime:** $O(\sqrt{m \log m})$ time per character.
A local tool for non-local problems

The cross-correlation between arrays $T$ and $P$ is defined by:

$$(T \otimes P)[i] = m - 1 \sum_{j=0}^{m-1} T[i + j]P[j]$$

Cross-correlations are an important tool for pattern matching.

Example:

$T : 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0$

$P : 1 \ 0 \ 1$

$$(T \otimes P)[0] = 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 = 1$$
A **local** tool for **non-local** problems

The cross-correlation between arrays $T$ and $P$ is defined by:

$$(T \otimes P)[i] = \sum_{j=0}^{m-1} T[i + j]P[j]$$

Cross-correlations are an important tool for pattern matching.

**Example:**

$T : 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0$

$P : 1 \ 0 \ 1$

$$(T \otimes P)[1] = 0 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 = 1$$
A local tool for non-local problems

The cross-correlation between arrays $T$ and $P$ is defined by:

$$(T \otimes P)[i] = \sum_{j=0}^{m-1} T[i+j]P[j]$$

Cross-correlations are an important tool for pattern matching.

Example:

$T : \begin{array}{cccc} 1 & 0 & 0 & 1 \end{array} \begin{array}{cccc} 0 & 1 & 1 & 0 \end{array} \begin{array}{cccc} 1 & 0 & 1 & 0 \end{array}$

$P : \begin{array}{cccc} 1 & 0 & 1 \end{array}$

$$(T \otimes P)[2] = 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 = 0$$
A **local** tool for **non-local** problems

The cross-correlation between arrays $T$ and $P$ is defined by:

$$(T \otimes P)[i] = \sum_{j=0}^{m-1} T[i + j]P[j]$$

*Cross-correlations are an important tool for pattern matching.*

Example:

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 0 0 1 0 1 1 0 1 0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 0 1</td>
<td></td>
</tr>
</tbody>
</table>

$$(T \otimes P)[3] = 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 = 2$$
A **local** tool for **non-local** problems

The cross-correlation between arrays $T$ and $P$ is defined by:

$$(T \otimes P)[i] = \sum_{j=0}^{m-1} T[i + j]P[j]$$

*The problem is local so we can apply the black box method.*

- Offline: $O(n \log m)$ total time (via FFTs).
- Pseudo-realtime: $O(\log^2 m)$ time per character.

**Method:** Peel apart your favourite pattern matching algorithm and replace the cross-correlation step.
Non-local results\footnote{C. S. CPM 2009}

<table>
<thead>
<tr>
<th>Problem</th>
<th>Offline per char time</th>
<th>Online/PsR penalty</th>
<th>Online Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Online</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Method: PsR Cross-correlations**

<table>
<thead>
<tr>
<th>Function</th>
<th>Offline per char time</th>
<th>Online/PsR penalty</th>
<th>Online Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>various</td>
<td>$O(\log m)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>self normalised</td>
<td>$O(\log m)$</td>
<td>$O(\log m)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>$L_2$ rearrangement</td>
<td>$O(\log m)$</td>
<td>$O(\log m)$</td>
<td>$O(m)$</td>
</tr>
</tbody>
</table>

**Method: Modified Blackbox**

<table>
<thead>
<tr>
<th>Swap-mismatch</th>
<th>Offline per char time</th>
<th>Online/PsR penalty</th>
<th>Online Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>swap-mismatch</td>
<td>$O(\sqrt{m\log m})$</td>
<td>$O(1)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>swap</td>
<td>$O(\log m \log</td>
<td>\Sigma_P</td>
<td>)$</td>
</tr>
<tr>
<td>overlap</td>
<td>$O(\log m)$</td>
<td>$O(\log m)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>k-diff with transpositions</td>
<td>$O(k)$</td>
<td>$O(\log m)$</td>
<td>$O(m)$</td>
</tr>
</tbody>
</table>

**Method: Other**

<table>
<thead>
<tr>
<th>$k$-differences</th>
<th>Offline per char time</th>
<th>Online/PsR penalty</th>
<th>Online Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-differences</td>
<td>$O(k)$</td>
<td>$O(1)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>edit distance/LCS</td>
<td>$O(m)$</td>
<td>$O(1)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>$L_1$ rearrangement</td>
<td>$O(m)$</td>
<td>$O(1)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>parameterised</td>
<td>$O(\log</td>
<td>\Sigma_P</td>
<td>)$</td>
</tr>
</tbody>
</table>
Can we improve the space complexity?

**Deterministically**, it has been shown that $\Omega(m)$ space is required to solve most *interesting* pattern matching problems.

Very recently, **randomised** algorithms\(^6\) were given for:

- Exact matching in $O(\log m)$ space and $O(\log m)$ time.
- $k$-mismatches in $O(k^3 \log m)$ space and $O(k^2 \log m)$ time.

\(^6\)Porat and Porat. FOCS 2009
Conclusions and Open Problems

Conclusions

- All *local* and most *non-local* pattern matching problems can be efficiently solved in the streaming model using $\Theta(m)$ space.
- Surprisingly, some problems such as Hamming distance and Swap-mismatch can be solved as efficiently online as offline.
- To move beyond $\Theta(m)$ space we need to randomise.

Open Problems

- Can we compute cross-correlations in $o(\log^2 m)$ time?
- What other pattern matching problems have $o(m)$ space randomised solutions? Or $\Omega(m)$ lower bounds?
- Is this the right model? What about multiple or unordered streams?