

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)

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 | a | ? | ? | ? | ? | ? | ? |
|---|---|---|---|---|---|---|---|---|---|

P :

| | | |
|---|---|---|
| a | b | a |
|---|---|---|

(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)

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 | a | a | ? | ? | ? | ? | ? |
|---|---|---|---|---|---|---|---|---|---|

P :

| | | |
|---|---|---|
| a | b | a |
|---|---|---|

(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)

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 | a | a | b | ? | ? | ? | ? |
|---|---|---|---|---|---|---|---|---|---|

P :

| | | |
|---|---|---|
| a | b | a |
|---|---|---|

 (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)

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 | a | a | b | a | ? | ? | ? |
|---|---|---|---|---|---|---|---|---|---|

P :

| | | |
|---|---|---|
| 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)

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 | a | a | b | a | b | ? | ? |
|---|---|---|---|---|---|---|---|---|---|

P :

| | | |
|---|---|---|
| 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)

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 | a | a | b | a | b | a | ? |
|---|---|---|---|---|---|---|---|---|---|

P:

| | | |
|---|---|---|
| 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)

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

P :

| | | |
|---|---|---|
| a | b | a |
|---|---|---|

 (dist = 3)

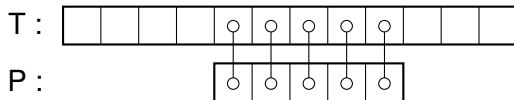
- ▶ Find the **distance**, $d(i)$, between $T[i, i + m - 1]$ and P for all i .
(Hamming distance shown)
- ▶ We are concerned with **worst-case** time per text character.
(Pseudo-Realtime)

Local and non-local pattern matching

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

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

Δ is some function acting on alphabet symbols)



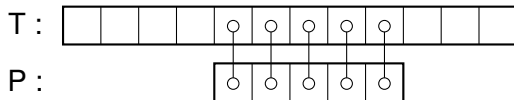
- ▶ Hamming, L_1 , L_2 , less-than and k-mismatch are all local.
- ▶ Edit distance and k-differences are non-local.

Local and non-local pattern matching

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

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

Δ is some function acting on alphabet symbols)



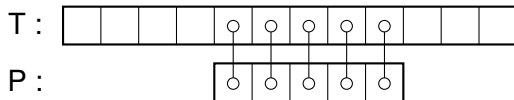
- ▶ Hamming, L_1 , L_2 , less-than and k-mismatch are all local.
- ▶ Edit distance and k-differences are non-local.

Local and non-local pattern matching

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

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

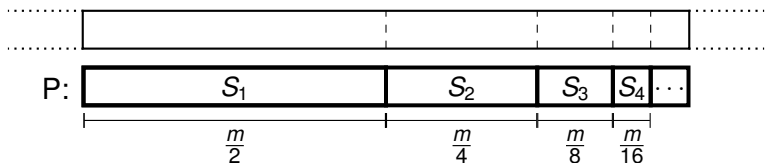
Δ is some function acting on alphabet symbols)



- ▶ Hamming, L_1 , L_2 , less-than and k-mismatch are all local.
- ▶ Edit distance and k-differences are non-local.

Local online pattern matching¹

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

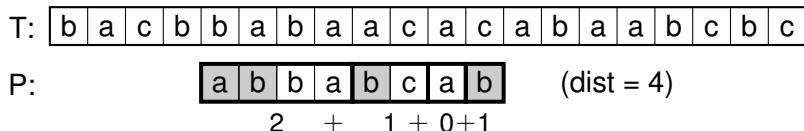


- ▶ Compute distances by summing the distance from each S_j to T .

¹C., Efremenko, Porat and Porat. CPM 2008

Local online pattern matching

Example: (using Hamming distance)

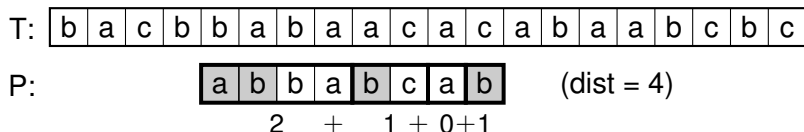


Plan:

- ▶ 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)



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



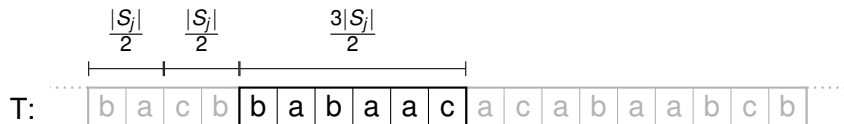
- ▶ Partition text into overlapping substrings for each S_j .
- ▶ Use offline algorithm as black box in each text partition.
- ▶ Distribute the work across the next $|S_j|/2$ characters.
- ▶ Time complexity increases by at worst multiplicative $O(\log m)$.

Local online pattern matching



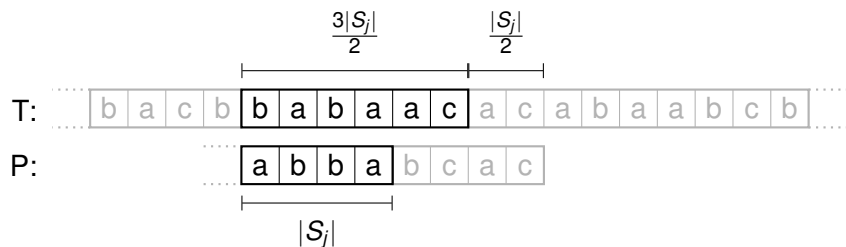
- ▶ Partition text into overlapping substrings for each S_j .
- ▶ Use offline algorithm as black box in each text partition.
- ▶ Distribute the work across the next $|S_j|/2$ characters.
- ▶ Time complexity increases by at worst multiplicative $O(\log m)$.

Local online pattern matching



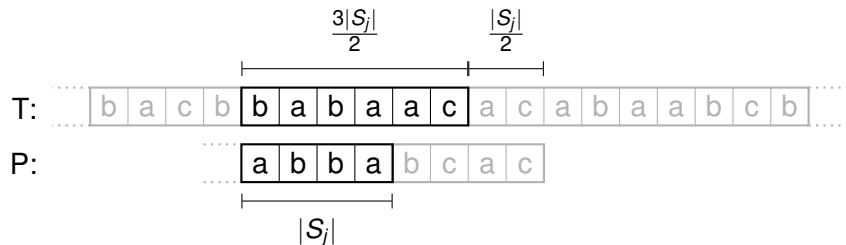
- ▶ Partition text into overlapping substrings for each S_j .
- ▶ Use offline algorithm as black box in each text partition.
- ▶ Distribute the work across the next $|S_j|/2$ characters.
- ▶ Time complexity increases by at worst multiplicative $O(\log m)$.

Local online pattern matching



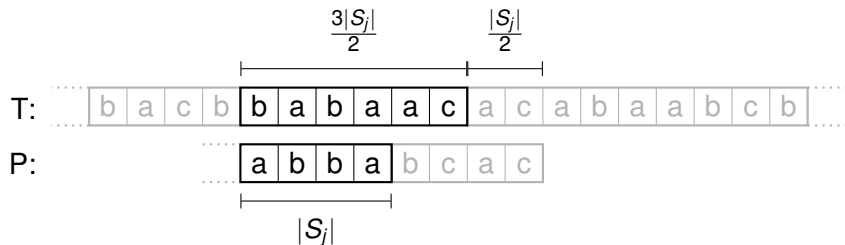
- ▶ Partition text into overlapping substrings for each S_j .
- ▶ Use offline algorithm as black box in each text partition.
- ▶ Distribute the work across the next $|S_j|/2$ characters.
- ▶ Time complexity increases by at worst multiplicative $O(\log m)$.

Local online pattern matching



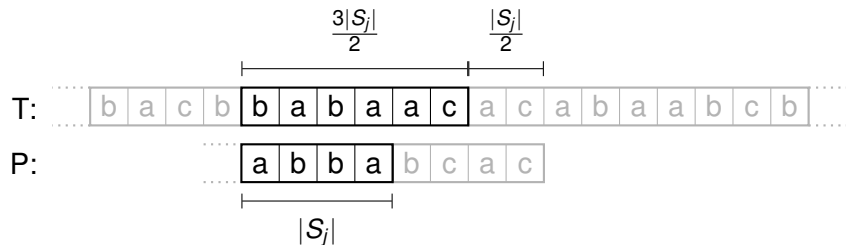
- ▶ Partition text into overlapping substrings for each S_j .
- ▶ Use offline algorithm as black box in each text partition.
- ▶ Distribute the work across the next $|S_j|/2$ characters.
- ▶ Time complexity increases by at worst multiplicative $O(\log m)$.

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 Results

| Problem | Offline per char time | Online penalty | Online space |
|-------------------------------|-----------------------|----------------|--------------|
| Exact matching with wildcards | $O(\log m)$ | $O(\log m)$ | $O(m)$ |
| Hamming distance | $O(\sqrt{m \log m})$ | $O(1)$ | $O(m)$ |
| k -mismatch | $O(\sqrt{k \log k})$ | $O(\log m)$ | $O(m)$ |
| L_1 distance | $O(\sqrt{m \log m})$ | $O(1)$ | $O(m)$ |
| L_2 distance | $O(\log m)$ | $O(\log m)$ | $O(m)$ |

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

| Problem | Offline per char time | Online penalty | Online space |
|-------------------------------|-----------------------|----------------|--------------|
| Exact matching with wildcards | $O(\log m)$ | $O(\log m)$ | $O(m)$ |
| Hamming distance | $O(\sqrt{m \log m})$ | $O(1)$ | $O(m)$ |
| k -mismatch | $O(\sqrt{k \log k})$ | $O(\log m)$ | $O(m)$ |
| L_1 distance | $O(\sqrt{m \log m})$ | $O(1)$ | $O(m)$ |
| L_2 distance | $O(\log m)$ | $O(\log m)$ | $O(m)$ |

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

| Problem | Offline per char time | Online penalty | Online space |
|-------------------------------|-----------------------|----------------|--------------|
| Exact matching with wildcards | $O(\log m)$ | $O(\log m)$ | $O(m)$ |
| Hamming distance | $O(\sqrt{m \log m})$ | $O(1)$ | $O(m)$ |
| k -mismatch | $O(\sqrt{k \log k})$ | $O(\log m)$ | $O(m)$ |
| L_1 distance | $O(\sqrt{m \log m})$ | $O(1)$ | $O(m)$ |
| L_2 distance | $O(\log m)$ | $O(\log m)$ | $O(m)$ |

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

| Problem | Offline per char time | Online penalty | Online space |
|-------------------------------|-----------------------|----------------|--------------|
| Exact matching with wildcards | $O(\log m)$ | $O(\log m)$ | $O(m)$ |
| Hamming distance | $O(\sqrt{m \log m})$ | $O(1)$ | $O(m)$ |
| k -mismatch | $O(\sqrt{k \log k})$ | $O(\log m)$ | $O(m)$ |
| L_1 distance | $O(\sqrt{m \log m})$ | $O(1)$ | $O(m)$ |
| L_2 distance | $O(\log m)$ | $O(\log m)$ | $O(m)$ |

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 |
|---|---|---|---|---|---|

 (dist = 2)

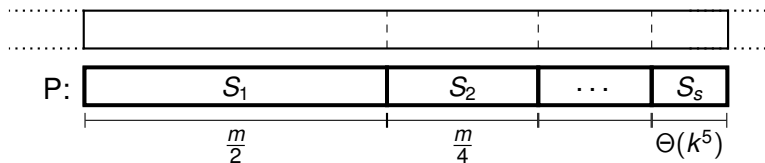
- ▶ Offline: $O(n\sqrt{k \log k})$ time².
- ▶ Pseudo-realtime: $O(\sqrt{k \log k} \log m)$ time per character.

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

²Amir, Lewenstein, Porat. SODA 2000

Can we improve the k -mismatch algorithm?

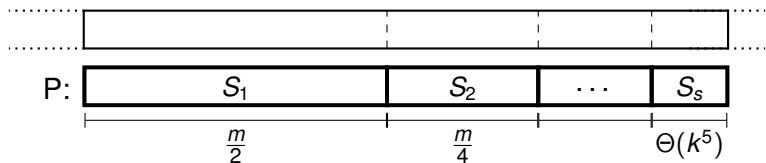
- ▶ Split the pattern into $s \in O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.
- ▶ However, we set s so that $k^5/2 \leq |S_s| < k^5$.



- ▶ 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?

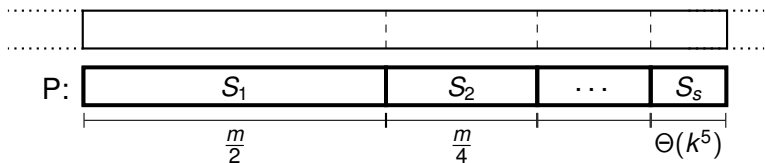
- ▶ Split the pattern into $s \in O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.
- ▶ However, we set s so that $k^5/2 \leq |S_s| < k^5$.



- ▶ 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?

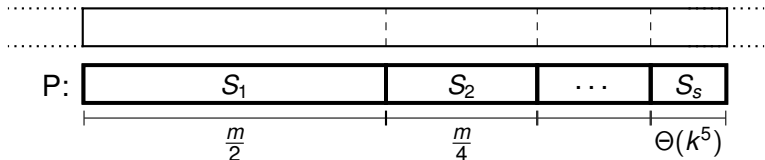
- ▶ Split the pattern into $s \in O(\log m)$ consecutive subpatterns where each subpattern is half the length of the previous.
- ▶ However, we set s so that $k^5/2 \leq |S_s| < k^5$.



- ▶ 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, \dots, S_{s-1} ?
- ▶ Each of these subpatterns has length greater than k^5 .



- ▶ For the case where $m \gg k$ there is an offline algorithm³ 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.

³Amir, 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 *realtime LCE processing* and *new filtering techniques*.
- ▶ Using this algorithm coupled with the above technique the time complexity can be further improved to

$$O(\sqrt{k} \log k + \log m)$$

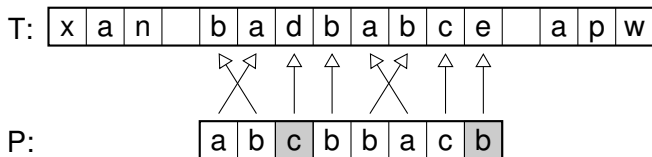
Open Problem: Remove the additive $O(\log m)$

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).

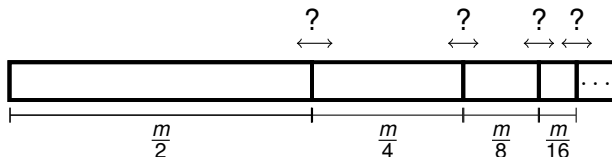


Offline: $O(n\sqrt{m \log m})$ time⁴.

⁴Amir, 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:

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

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 :

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|

P :

| | | |
|---|---|---|
| 1 | 0 | 1 |
|---|---|---|

$$(T \otimes P)[2] = 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 = 0$$

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:

T :

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|

P :

| | | |
|---|---|---|
| 1 | 0 | 1 |
|---|---|---|

$$(T \otimes P)[3] = 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 = 2$$

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.

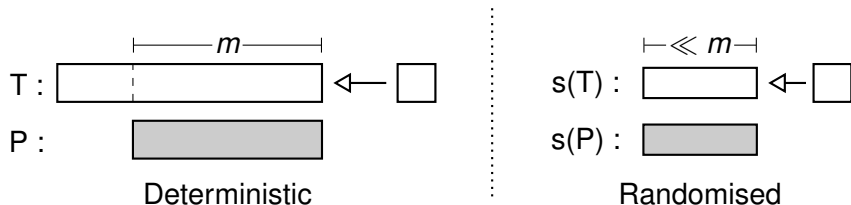
Non-local results⁵

| Problem | Offline per char time | Online/PsR penalty | Online Space |
|--|-----------------------------|--------------------------|--------------------------|
| <i>Method : PsR Cross-correlations</i> | | | |
| function | various | $O(\log m)$ | $O(m)$ |
| self normalised | $O(\log m)$ | $O(\log m)$ | $O(m)$ |
| L_2 rearrangement | $O(\log m)$ | $O(\log m)$ | $O(m)$ |
| <i>Method : Modified Blackbox</i> | | | |
| swap-mismatch | $O(\sqrt{m \log m})$ | $O(1)$ | $O(m)$ |
| swap | $O(\log m \log \Sigma_P)$ | $O(\log m)$ | $O(m)$ |
| overlap | $O(\log m)$ | $O(\log m)$ | $O(m)$ |
| k-diff with transpositions | $O(k)$ | $O(\log m)$ | $O(m)$ |
| <i>Method: Other</i> | | | |
| k-differences | $O(k)$ | $O(1)$ | $O(m)$ |
| edit distance/LCS | $O(m)$ | $O(1)$ | $O(m)$ |
| L_1 rearrangement | $O(m)$ | $O(1)$ | $O(m)$ |
| parameterised | $O(\log \Sigma_P)$ | $O(1)$ | $O(m)$ |

⁵C. S. CPM 2009

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** algorithms⁶ were given for:

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

⁶Porat 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?