Streaming and property testing algorithms for string processing

Tatiana Starikovskaya

Based on joint work with:
R. Clifford, P. Gawrychowski, A. Fontaine, E. Porat, B. Sach

- Pattern matching has been studied for **40+ years**
- **More than 85 algorithms**
- KMP algorithm uses $O(|P|)$ space and $O(|T|)$ time, and Aho-Corasick achieves similar bounds for dictionary matching
- **We can't do better**: we must store a description of the pattern(s) and we must read the whole text

GAME OVER

Intrusion Detection Systems

- Large number of patterns
- Search patterns represent portions of known attack patterns and have length 1 – 30
- If only cache memory is used, the algorithm can benefit most from a high performance cache

Outline of today's talk

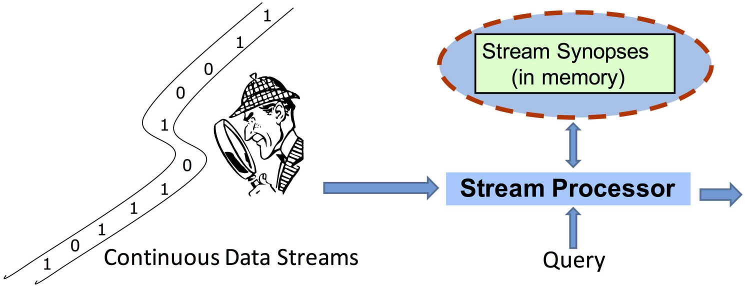
Streaming model

- Exact pattern matching
- Approximate pattern matching (Hamming distance)
- Approximate pattern matching (edit distance)
- Preprocessing

Property testing model

- Exact pattern matching

Streaming model



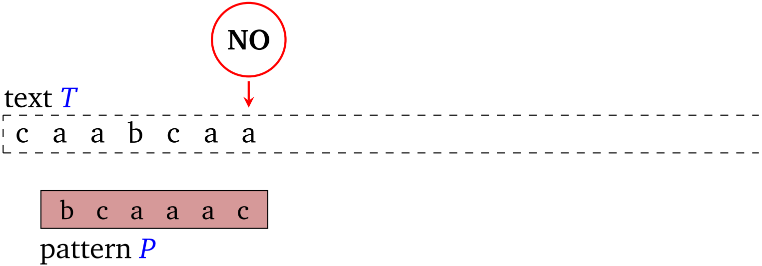
We want to process the stream **on-the-fly & in small space**

Part I: Exact pattern matching

Exact pattern matching

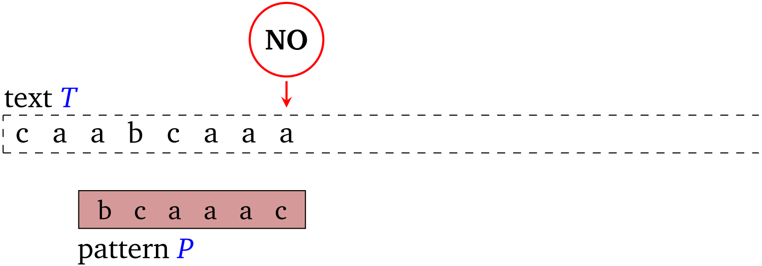
- **Query** = "Is there an occurrence of P?"
- **Space** = total space used by the stream processor
- **Time** = time per position of T

Exact pattern matching



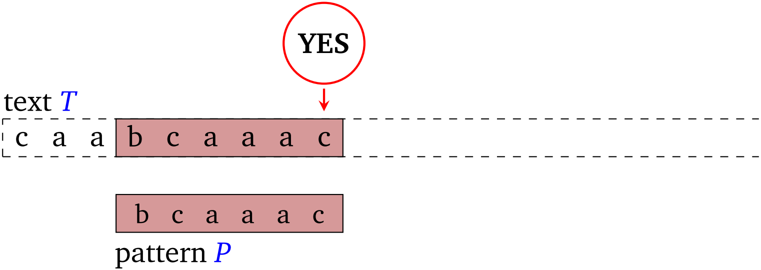
- **Query** = "Is there an occurrence of P?"
- **Space** = total space used by the stream processor
- **Time** = time per position of T

Exact pattern matching



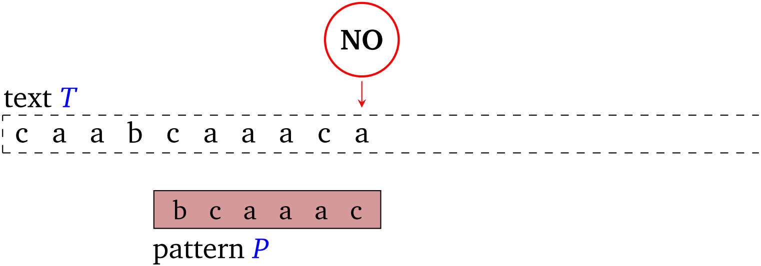
- **Query** = "Is there an occurrence of P?"
- **Space** = total space used by the stream processor
- **Time** = time per position of T

Exact pattern matching



- **Query** = "Is there an occurrence of P?"
- **Space** = total space used by the stream processor
- **Time** = time per position of T

Exact pattern matching



- **Query** = "Is there an occurrence of P?"
- **Space** = total space used by the stream processor
- **Time** = time per position of T

Karp-Rabin algorithm

Karp-Rabin fingerprint

$$\varphi(s_1 s_2 \ldots s_m) = \sum_{i=1}^{m} s_i r^{m-i} \bmod p$$

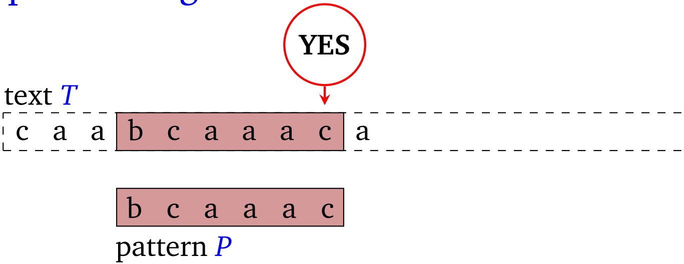
where p is a prime and r is a random integer $\in [0, p-1]$

It's a good hash function

S_1, S_2 are two strings of length m, the prime p is large

1. $S_1 = S_2 \Rightarrow \varphi(S_1) = \varphi(S_2)$
2. $S_1 \neq S_2$, lengths of S_1, S_2 are equal $\Rightarrow \varphi(S_1) \neq \varphi(S_2)$ w.h.p.

Karp-Rabin algorithm



When a new character $t_i = a$ arrives:

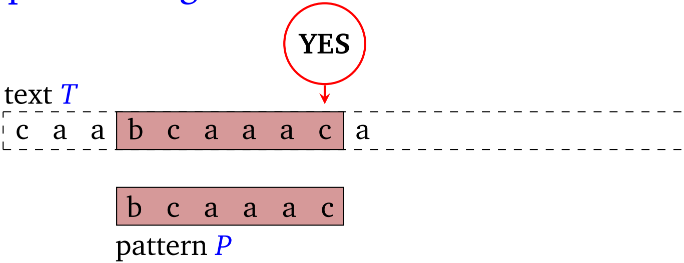
1. Compute the fingerprint $\varphi(t_{i-m+1} \dots t_{i-1} t_i)$ in $O(1)$ time

$\varphi(\underline{caaac}c) = \Big((\varphi(b\underline{caaac}) - br^{m-1}) \cdot r + a \bmod p$

2. If $\varphi(t_{i-m+1} \dots t_{i-1} t_i) = \varphi(P)$, output "YES"

We need t_{i-m} to update the fingerprint $\Rightarrow$ **we must store** $t_{i-m}, \dots, t_{i-1}$

Karp-Rabin algorithm



K.-R. algorithm is a **streaming pattern matching algorithm** that uses $\Theta(m)$ space and $O(1)$ time per character of T

It finds all occurrences of P in T correctly w.h.p.

Exact pattern matching

Authors	Space [1]	Time
Single pattern		
Karp & Rabin, 1987	$\Theta(m)$	$O(1)$
Porat & Porat, 2009	$O(\log m)$	$O(\log m)$
Breslauer & Galil, 2011	$O(\log m)$	$O(1)$

Dictionary of d patterns		
Clifford, Fontaine, Porat Sach, S., 2015	$O(d \log m)$	$O(\log\log(m+d))$
Golan & Porat, 2017	$O(d \log m)$ $O(\lvert\Sigma\rvert^{\varepsilon} d \log(m/\varepsilon))$	$O(\log\log\lvert\Sigma\rvert)$ $O(1/\varepsilon)$

[1] In words

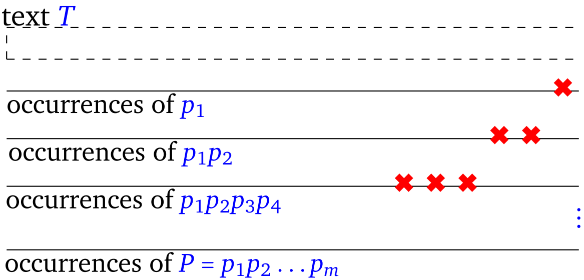
Exact pattern matching

Authors	Space [1]	Time
Single pattern		
Karp & Rabin, 1987	$\Theta(m)$	$O(1)$
Porat & Porat, 2009 ★	$O(\log m)$	$O(\log m)$
Breslauer & Galil, 2011	$O(\log m)$	$O(1)$

Dictionary of d patterns		
Clifford, Fontaine, Porat Sach, S., 2015	$O(d \log m)$	$O(\log\log(m+d))$
Golan & Porat, 2017	$O(d \log m)$ $O(\|\Sigma\|^{\varepsilon} d \log(m/\varepsilon))$	$O(\log\log\|\Sigma\|)$ $O(1/\varepsilon)$

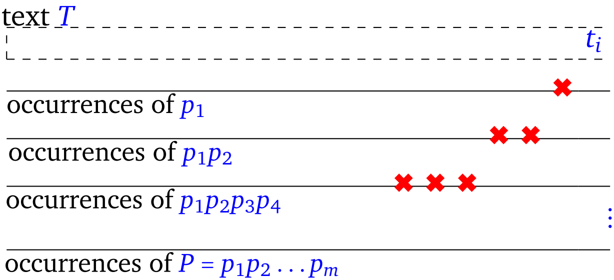
[1] In words

Porat & Porat, 2009 ★



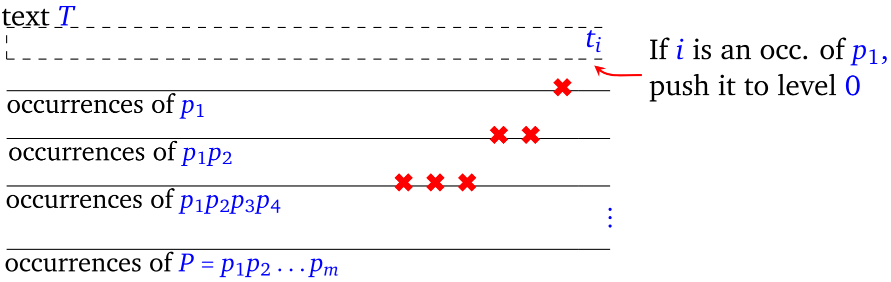
for each character t_i **do**
  if $t_i = p_1$ **then** push i to level 0
  for each $j = 0, \dots, \log m - 1$
    $lp \leftarrow$ leftmost position in level j
    if $i - lp + 1 = 2^{j+1}$ **then**
      Pop lp from level j
      if $\varphi(t_{lp} \dots t_i) = \varphi(p_1 \dots p_{2^{j+1}})$ **then** push lp to level $j + 1$

Porat & Porat, 2009 ★



for each character t_i **do**
  if $t_i = p_1$ **then** push i to level 0
  for each $j = 0, \dots, \log m - 1$
    $lp \leftarrow$ leftmost position in level j
    if $i - lp + 1 = 2^{j+1}$ **then**
      Pop lp from level j
      if $\varphi(t_{lp} \dots t_i) = \varphi(p_1 \dots p_{2^{j+1}})$ **then** push lp to level $j+1$

Porat & Porat, 2009 ★



for each character t_i **do**

  if $t_i = p_1$ **then** push i to level 0

  for each $j = 0, \dots, \log m - 1$

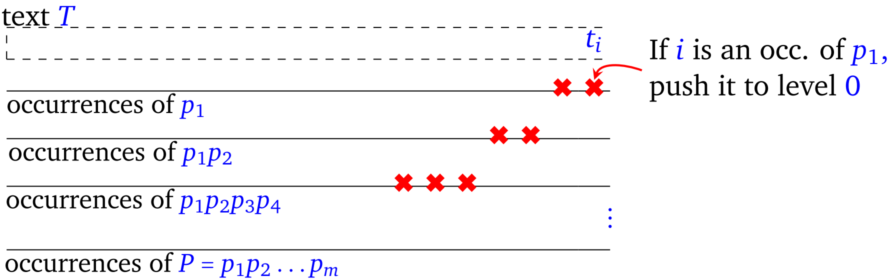
    $lp \leftarrow$ leftmost position in level j

    if $i - lp + 1 = 2^{j+1}$ **then**

      Pop lp from level j

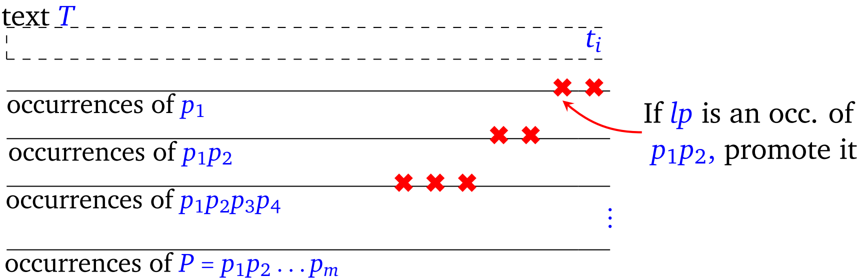
      if $\varphi(t_{lp} \dots t_i) = \varphi(p_1 \dots p_{2^{j+1}})$ **then** push lp to level $j + 1$

Porat & Porat, 2009 ★



for each character t_i **do**
  if $t_i = p_1$ **then** push i to level 0
  for each $j = 0, \ldots, \log m - 1$
    $lp \leftarrow$ leftmost position in level j
    if $i - lp + 1 = 2^{j+1}$ **then**
      Pop lp from level j
      if $\varphi(t_{lp} \ldots t_i) = \varphi(p_1 \ldots p_{2^{j+1}})$ **then** push lp to level $j + 1$

Porat & Porat, 2009 ★



for each character t_i **do**

  if $t_i = p_1$ **then** push i to level 0

  for each $j = 0, \ldots, \log m - 1$

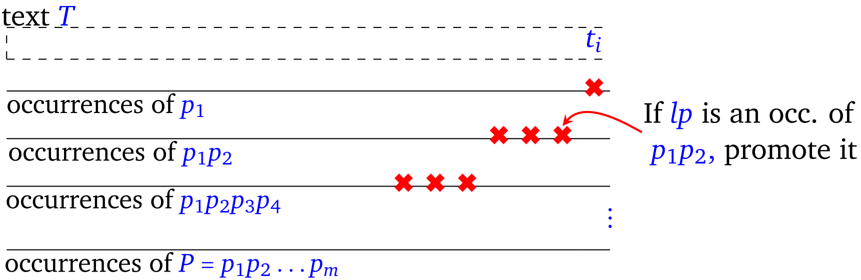
    $lp \leftarrow$ leftmost position in level j

    if $i - lp + 1 = 2^{j+1}$ **then**

      Pop lp from level j

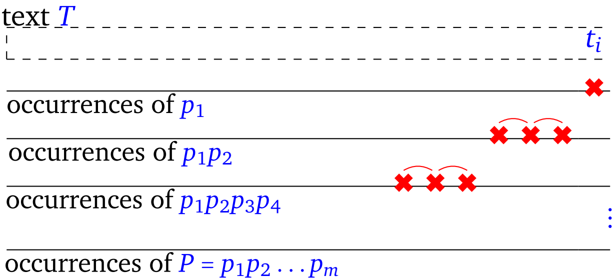
      if $\varphi(t_{lp} \ldots t_i) = \varphi(p_1 \ldots p_{2^{j+1}})$ **then** push lp to level $j + 1$

Porat & Porat, 2009 ★



for each character t_i **do**
    if $t_i = p_1$ **then** push i to level 0
    for each $j = 0, \ldots, \log m - 1$
        $lp \leftarrow$ leftmost position in level j
        if $i - lp + 1 = 2^{j+1}$ **then**
            Pop lp from level j
            if $\varphi(t_{lp} \ldots t_i) = \varphi(p_1 \ldots p_{2^{j+1}})$ **then** push lp to level $j + 1$

Porat & Porat, 2009 ★

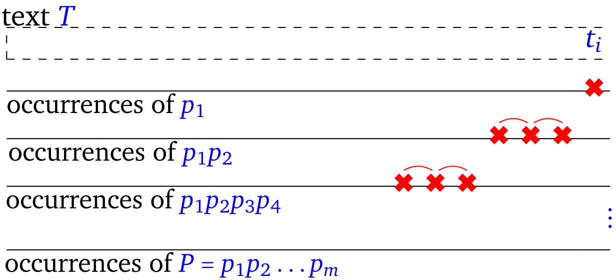


Lemma If there are ≥ 3 occurrences of a 2^j-length string in a 2^{j+1}-length string, the occurrences form a run

For each level we store:

- The leftmost and the second leftmost positions lp, lp'
- The fingerprints of $t_1t_2\dots t_{lp}$, $t_{lp+1}\dots t_{lp'}$, and $t_1\dots t_i$

Porat & Porat, 2009 ★



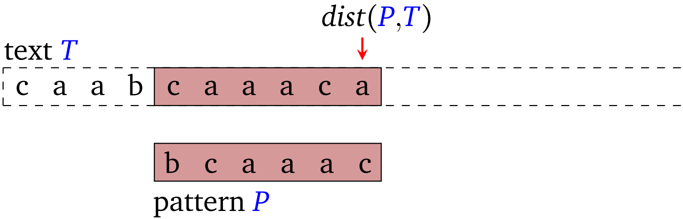
For each level we need:

- $O(1)$ space
- $O(1)$ time for updating and extracting $\varphi(t_{lp} \dots t_i)$

Theorem Porat & Porat algorithm is a streaming pattern matching algorithm that uses $O(\log m)$ space and $O(\log m)$ time per character

Part II: Approximate pattern matching

Approximate pattern matching

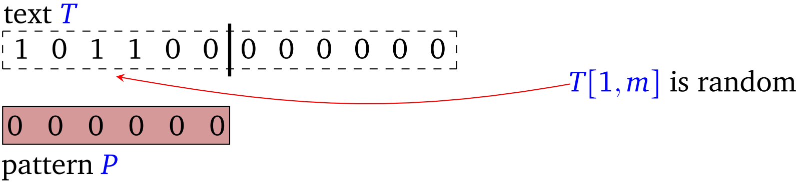


- **Query** = “Distance between P and T”
- **Distance:** Hamming, edit, ...

Approximate pattern matching (Hamming distance)

Any streaming algorithm for computing **exact** Hamming distances must use $\Omega(m)$ space

By **Yao's minimax principle** it suffices to consider deterministic algorithms on "hard" distribution of the inputs



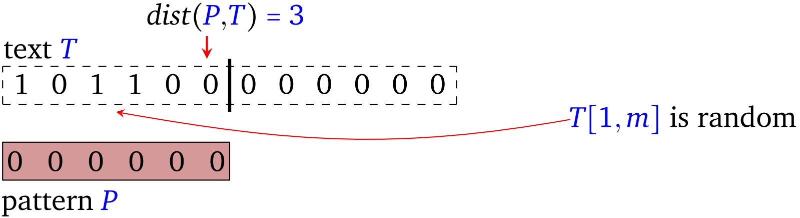
After reading $T[m]$, the algorithm cannot go back and read one of the letters $T[1], T[2], \ldots, T[m]$, but can restore $T[1, m]$

Therefore, it stores a full description of $T[1, m] \Rightarrow \Omega(m)$ space by information-theoretic ideas

Approximate pattern matching (Hamming distance)

Any streaming algorithm for computing **exact** Hamming distances must use $\Omega(m)$ space

By **Yao's minimax principle** it suffices to consider deterministic algorithms on "hard" distribution of the inputs



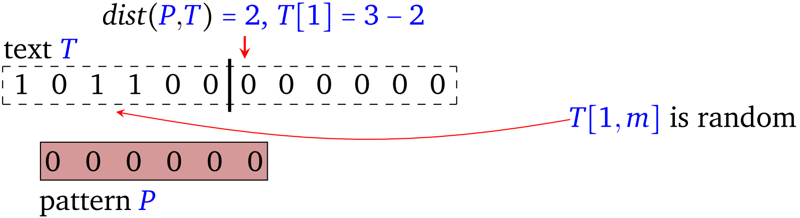
After reading $T[m]$, the algorithm cannot go back and read one of the letters $T[1], T[2], \ldots, T[m]$, but can restore $T[1, m]$

Therefore, it stores a full description of $T[1, m] \Rightarrow \Omega(m)$ space by information-theoretic ideas

Approximate pattern matching (Hamming distance)

Any streaming algorithm for computing **exact** Hamming distances must use $\Omega(m)$ space

By **Yao's minimax principle** it suffices to consider deterministic algorithms on "hard" distribution of the inputs



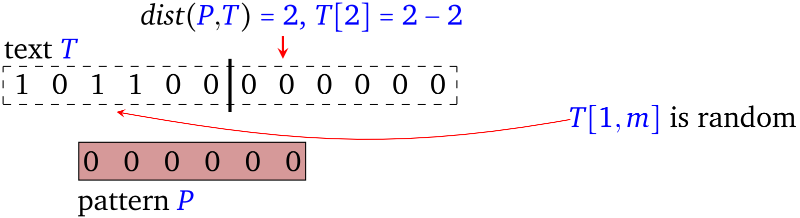
After reading $T[m]$, the algorithm cannot go back and read one of the letters $T[1], T[2], \ldots, T[m]$, but can restore $T[1,m]$

Therefore, it stores a full description of $T[1,m] \Rightarrow \Omega(m)$ space by information-theoretic ideas

Approximate pattern matching (Hamming distance)

Any streaming algorithm for computing **exact** Hamming distances must use $\Omega(m)$ space

By **Yao's minimax principle** it suffices to consider deterministic algorithms on "hard" distribution of the inputs



After reading $T[m]$, the algorithm cannot go back and read one of the letters $T[1], T[2], \ldots, T[m]$, but can restore $T[1,m]$

Therefore, it stores a full description of $T[1,m] \Rightarrow \Omega(m)$ space by information-theoretic ideas

Approximate pattern matching (Hamming distance)

Authors	Space [2]	Time
Single pattern, only distances $\leq k$		
Porat & Porat, 2009	$\tilde{O}(k^3)$	$\tilde{O}(k^2)$
Clifford, Fontaine, Porat, Sach, S., 2016	$\tilde{O}(k^2)$	$\tilde{O}(\sqrt{k})$
Clifford, Kociumaka, Porat, 2018	$O(k \log \frac{m}{k})$	$O(k \log^3 m \log \frac{m}{k})$

Single pattern, $(1+\varepsilon)$-approx.		
Clifford, S., 2016	$O(\varepsilon^{-5}\sqrt{m}\log^4 m)$	$O(\varepsilon^{-4}\log^3 m)$

[2]In words

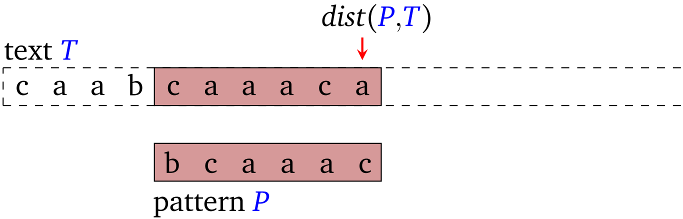
Approximate pattern matching (Hamming distance)

Authors	Space [2]	Time
Single pattern, only distances $\leq k$		
Porat & Porat, 2009 ★	$\tilde{O}(k^3)$	$\tilde{O}(k^2)$
Clifford, Fontaine, Porat, Sach, S., 2016	$\tilde{O}(k^2)$	$\tilde{O}(\sqrt{k})$
Clifford, Kociumaka, Porat, 2018	$O(k \log \frac{m}{k})$	$O(k \log^3 m \log \frac{m}{k})$

Single pattern, $(1+\varepsilon)$-approx.		
Clifford, S., 2016	$O(\varepsilon^{-5}\sqrt{m}\log^4 m)$	$O(\varepsilon^{-4}\log^3 m)$

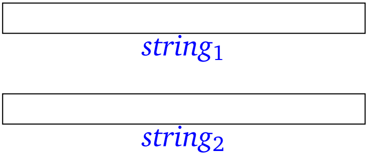
[2] In words

Porat & Porat, 2009 ★



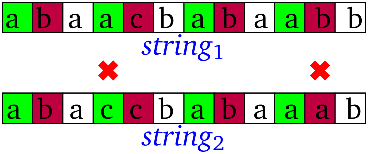
- If HAM(P,T) $> k$, output “NO”
- Otherwise, output HAM(P,T)

From 1 mismatch to exact pattern matching



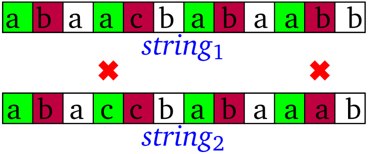
- Is HAM $(string_1, string_2) = 1$?

From 1 mismatch to exact pattern matching



- Is $\mathrm{HAM}(string_1, string_2) = 1$?
- Partition the strings into substrings of q colors
- One mismatch $\Rightarrow$ one pair of substrings does not match
- **Hope:** If there are ≥ 2 mismatches, they will end up in substrings of different colors $\Rightarrow$ at least 2 pairs of substrings do not match

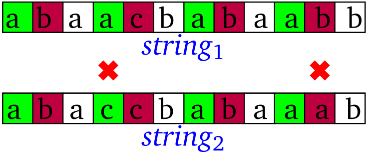
From 1 mismatch to exact pattern matching



For each prime $q \in [\log m, \log^2 m]$:
  Partition $string_1$ into q equi-spaced substrings
  Partition $string_2$ into q equi-spaced substrings

In total: $O(\log m)$ primes, and for each prime there are $O(\log^2 m)$ pairs of substrings

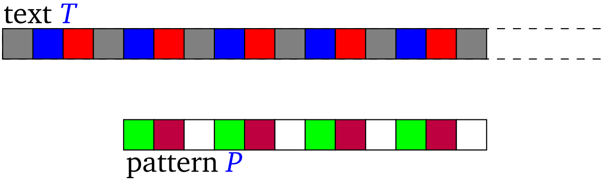
From 1 mismatch to exact pattern matching



Lemma There are ≥ 2 mismatches ✖$_1$, ✖$_2$ $\Rightarrow$ there exists a prime q such that at least two pairs of substrings do not match

- ✖$_1$, ✖$_2$ in the same pair $\Leftrightarrow$ ✖$_1$ − ✖$_2$ $= 0 \pmod{q}$
- $m \geq$ ✖$_1$ − ✖$_2$ cannot be a multiple of $\log m$ distinct primes

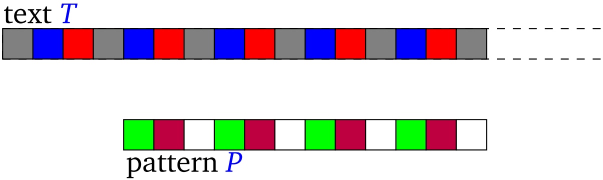
From 1 mismatch to exact pattern matching



Is HAM(P, T) = 1?

for each position of the text T **do**
  for each prime q in $[\log m, \log^2 m]$ **do**
    $h \leftarrow$ number of (substream, subpattern) that mismatch
    if $h = 0$ **OR** $h > 1$ **return** "NO"
  return "YES"

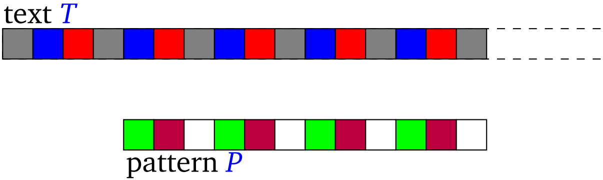
From 1 mismatch to exact pattern matching



Compute number of mismatching pairs

for each prime q in $[\log m, \log^2 m]$ **do**
  for each (substream, subpattern) **do**
    run streaming exact pattern matching

From 1 mismatch to exact pattern matching



Complexity

$$\text{Space} = O(\underbrace{\log m}_{\#\text{ of primes}} \cdot \underbrace{\log^2 m}_{\#\text{ of substr.}} \cdot \underbrace{\log^2 m}_{\#\text{ of subpatterns}} \cdot \log m)$$

$$\text{Time} = O(\underbrace{\log m}_{\#\text{ of primes}} \cdot \underbrace{\log^2 m}_{\#\text{ of substr.}} \cdot \underbrace{\log^2 m}_{\#\text{ of subpatterns}})$$

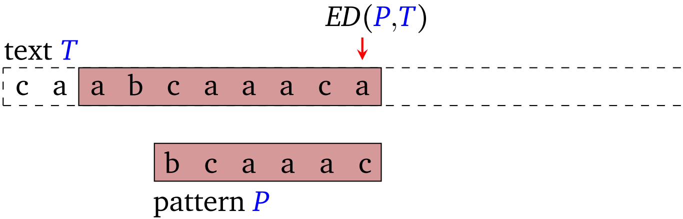
Approximate pattern matching (Hamming distance)

Porat & Porat, 2009
$\tilde{O}(k^3)$ space, $\tilde{O}(k^2)$ time
Same as for $k = 1$ but take more primes

Clifford, Fontaine, Porat, Sach, S., 2016
$\tilde{O}(k^2)$ space, $\tilde{O}(\sqrt{k})$ time
We can take fewer primes if we choose them at random + periodicity to improve time

Clifford, Kociumaka, Porat, 2018
$O(k \log \frac{m}{k})$ space, $O(k \log^3 m \log \frac{m}{k})$ time
New encoding for mismatch information + periodicity + exponentially growing prefixes

Approximate pattern matching (edit distance)

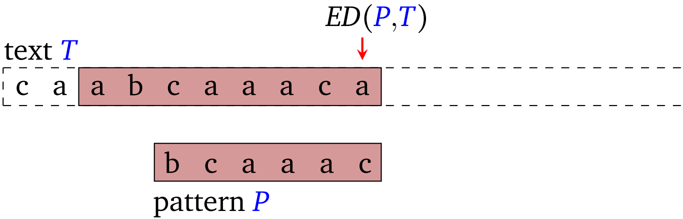


$ED(P,S)$ = minimum number of insertions, deletions, and replacements that transform P into S

Example: P = aaac, S = abacb, edit distance = 2

- If $ED(P,T) > k$, output "NO"
- Otherwise, output $ED(P,T)$

Approximate pattern matching (edit distance)



$ED(P,S)$ = minimum number of insertions, deletions, and replacements that transform P into S

Example: P = aaac, S = abacb, edit distance = 2

- Hybrid dynamic programming: $\mathcal{O}(m)$ space, $\mathcal{O}(k)$ time
- S., 2017: $\mathcal{O}(\sqrt{m} \cdot poly(k, \log m))$ space, $\mathcal{O}(\sqrt{m} \cdot poly(k, \log m))$ time

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8	…	$3n$
0	0	1	1	0	1	1	0	0	…	0
1	1	1	1	1	0	1	0	1	…	1

Copy letters of S to S':

	1	2	3		n
S :	**0**	1	0	…	0
S' :					

text position = 1, j = 1

1. Copy $S[i]$. If $h_j(S[i]) = 1$, move to the right;
2. $j = j + 1$.

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8		$3n$
0	0	1	1	0	1	1	0	0	…	0
1	1	1	1	1	0	1	0	1	…	1

Copy letters of S to S':

	1	2	3		n
S :	**0**	1	0	…	0
S' :	0				

text position = 1, j = 1

1. Copy $S[i]$. If $h_j(S[i]) = 1$, move to the right;
2. $j = j + 1$.

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8	…	$3n$
0	**0**	1	1	0	1	1	0	0	…	0
1	1	1	1	1	0	1	0	1	…	1

Copy letters of S to S':

	1	2	3	…	n
$S:$	**0**	1	0	…	0
$S':$	0				

text position = 1, j = 1

1. Copy $S[i]$. If $h_j(S[i]) = 1$, move to the right;
2. $j = j + 1$.

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8		$3n$
0	0	1	1	0	1	1	0	0	…	0
1	1	1	1	1	0	1	0	1	…	1

Copy letters of S to S':

	1	2	3		n
S :	**0**	1	0	…	0
S' :	0				

text position = 1, j = 2

1. Copy $S[i]$. If $h_j(S[i]) = 1$, move to the right;
2. $j = j + 1$.

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8		…	$3n$
0	0	1	1	0	1	1	0	0		…	0
1	1	1	1	1	0	1	0	1		…	1

Copy letters of S to S':

	1	2	3		n
S :	**0**	1	0	…	0
S' :	0	0			

text position = 1, j = 2

1. Copy $S[i]$. If $h_j(S[i]) = 1$, move to the right;
2. $j = j + 1$.

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8		3n
0	0	**1**	1	0	1	1	0	0	…	0
1	1	1	1	1	0	1	0	1	…	1

Copy letters of S to S':

	1	2	3		n
S :	**0**	1	0	…	0
S' :	0	0			

text position = 1, j = 2

1. Copy $S[i]$. If $h_j(S[i]) = 1$, move to the right;
2. $j = j + 1$.

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8	…	$3n$
0	0	1	1	0	1	1	0	0	…	0
1	1	1	1	1	0	1	0	1	…	1

Copy letters of S to S':

	1	2	3	…	n
S :	0	**1**	0	…	0
S' :	0	0			

text position $= 2, j = 3$

1. Copy $S[i]$. If $h_j(S[i]) = 1$, move to the right;
2. $j = j + 1$.

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8	…	3n
0	0	1	1	0	1	1	0	0	…	0
1	1	1	1	1	0	1	0	1	…	1

Copy letters of S to S':

	1	2	3	…	n
S :	0	**1**	0	…	0
S' :	0	0	1		

text position = 2, j = 3

1. Copy $S[i]$. If $h_j(S[i]) = 1$, move to the right;
2. $j = j + 1$.

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8	…	3n
0	0	1	1	0	1	1	0	0	…	0
1	1	**1**	1	1	0	1	0	1	…	1

Copy letters of S to S':

	1	2	3	…	n
S :	0	**1**	0	…	0
S' :	0	0	1		

text position = 2, j = 3

1. Copy $S[i]$. If $h_j(S[i]) = 1$, move to the right;
2. $j = j + 1$.

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8	…	$3n$
0	0	1	1	0	1	1	0	0	…	0
1	1	1	1	1	0	1	0	1	…	1

Copy letters of S to S':

	1	2	3	…	n
S :	0	1	0	…	0
S' :	0	0	1	…	

text position = 2, j = 3

If $ED(S,T) = k$, then $k/2 \leq HD(S',T') \leq \mathcal{O}(k^2)$ w/ prob. 0.99

Embedding from edit to Hamming distance

Chakraborty, Goldenberg, Koucky, 2016

Pick $3n$ random functions $h_j : \{0,1\} \to \{0,1\}$

	1	2	3	4	5	6	7	8		$3n$
0	0	1	1	0	1	1	0	0	…	0
1	1	1	1	1	0	1	0	1	…	1

Copy letters of S to S':

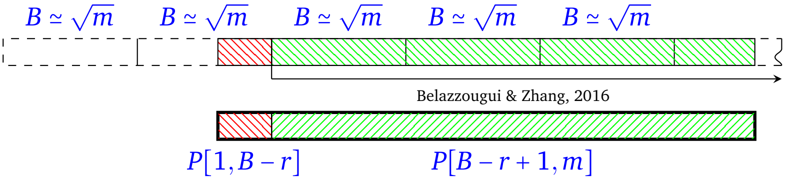
	1	2	3		n
S :	0	1	0	…	0
S' :	0	0	1	…	

text position = 2, j = 3

Belazzougui, Zhang, 2016

- Embedding + streaming alg'm for k^2-mismatch $\Rightarrow$ a good estimate for edit distance
- If $ED(S,T) \leq k$, $\tilde{O}(k^2)$ embeddings + streaming alg'm for k^2-mismatch $\Rightarrow$ **exact value!**

Approximate pattern matching (edit distance)



Starting from each block i, run Belazzougui & Zhang, 2016

$$ED[j] = \min_{i \in [r-k, r+k]} ED\big(P[1, B - i], \mathbf{T_1}\big) + ED\big(P[B - i + 1, m], \mathbf{T_2}\big)$$

We compute $ED\big(P[1, B - i], \mathbf{T_1}\big)$ while reading $\mathbf{T_1}$ using dynamic programming, then encode the distances to restore later

Part III: Preprocessing

Preprocessing for pattern matching

Can we preprocess the patterns in a streaming way?
If yes, do we need to read them several times?
How much space do we need?

Periodicity — Ergün, Jowhari, Saglam, 2010

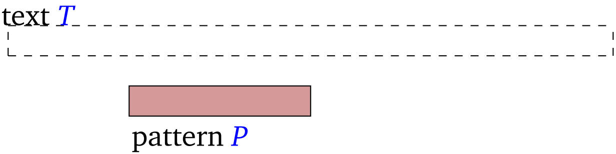
- Periodic patterns: $O(\log m)$ space, $O(\log m)$ time
- Non-periodic patterns: $\Omega(m)$ space
- 2 passes (periodic and non-periodic patterns): $O(\log m)$ space, $O(\log m)$ time

Periodicity with mismatches — Ergün et al., 2017

- Periodic patterns: $O(k^4 \log^9 n)$ space
- 2-pass algorithm for non-periodic patterns, lower bounds

Part IV: Property testing model

Pattern matching



Is T **free** from occurrences of P?

Same question when T and P are of dimension $d \geq 2$

Property testing model

If Sherlock wants to solve the problem fast, he can only query a few characters of T

Property testing model

Task: develop an ultra-efficient randomised algorithm to decide whether T is free from occurrences of P

We must

- accept, if T is ε_1-close to being P-free
- reject, if T is ε_2-far from being P-free
- accept or reject otherwise

ε_1-close = we can fix $\leq \varepsilon_1 n$ characters of T so that the property is satisfied

ε_2-far = we must fix $\geq \varepsilon_2 n$ characters of T so that the property is satisfied

Property testing model

Task: develop an ultra-efficient randomised algorithm to decide whether T is free from occurrences of P

We must

- accept, if T is ε_1-close to being P-free
- reject, if T is ε_2-far from being P-free
- accept or reject otherwise

Ben-Eliezer, Korman, Reichman, 2017

There is an algorithm which queries $O(\varepsilon^{-1})$ letters of T and distinguishes between $\varepsilon/2$-close and ε-far (for almost all patterns)

Summary of today's talk

It's all about **pattern matching**

Randomisation and approximation $\Rightarrow$ more efficient algorithms

Many open questions

Thank you!