APPROXIMATE STRING MATCHING -

The Hamming Distance Case

Amihood Amir

Bar-Ilan Univ.
&
Georgia Tech

Moshe Lewenstein

Ely Porat

Bar-Ilan Univ.

EXACT MATCHING

INPUT: Text $T = t_1 t_2 \cdots t_n$

Pattern $P = p_1 p_2 \cdots p_m$

$t_i, p_j \in \Sigma$

FIND: All text locations i where

$t_i = p_1$

$t_{i+1} = p_2$

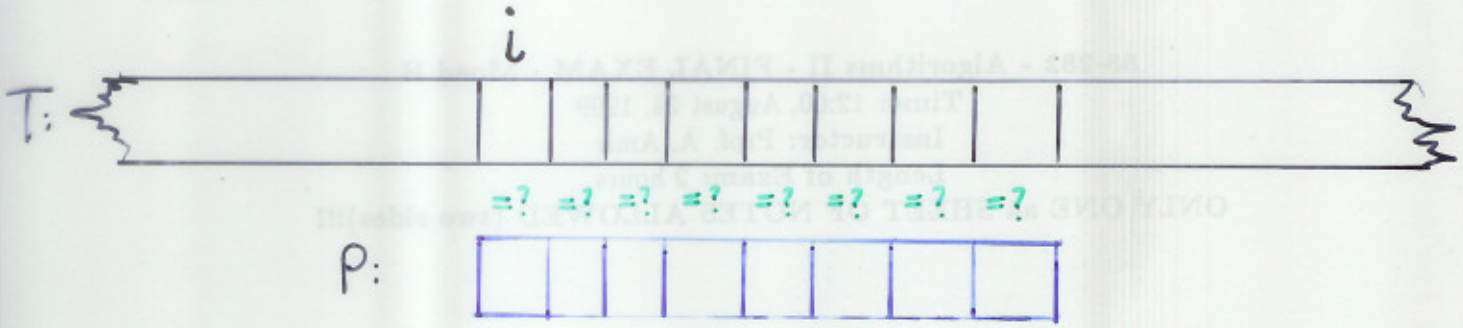
$t_{i+2} = p_3$

$\vdots$

$t_{i+m-1} = p_m$

Naive Algorithm:

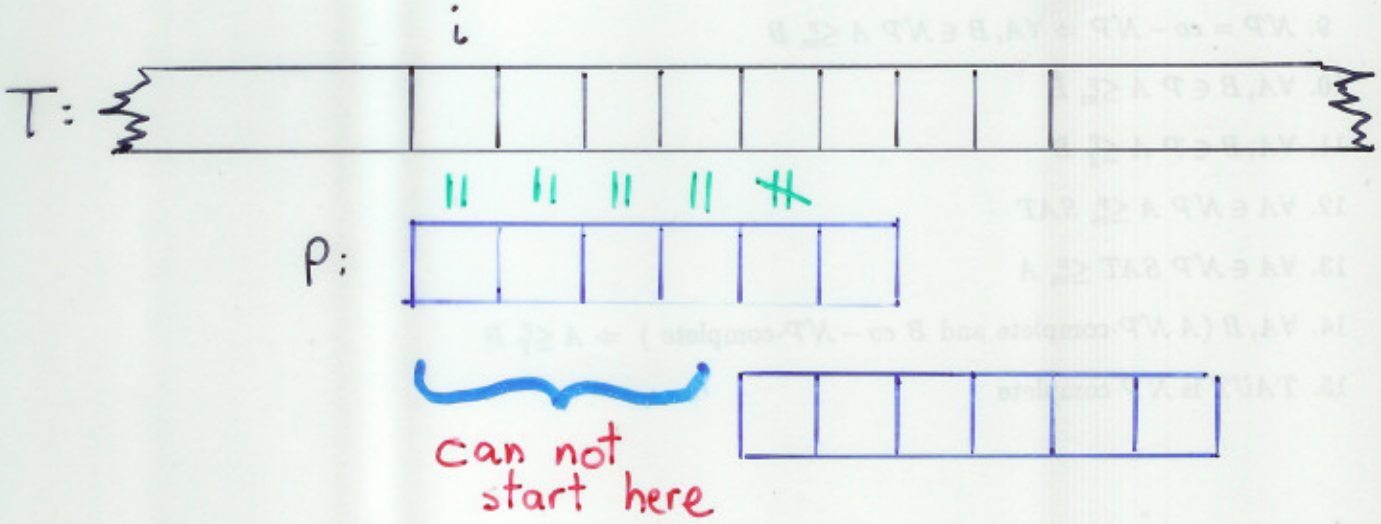
For every text location i:



Time: $O(nm)$

Special Case: Every pattern symbol different

(P = ABCDEFGH)



Time: $O(n)$

MOTIVATIONS FOR PATTERN MATCHING

Text Editors

Molecular Biology

Computer Vision

Meteorology

⋮

For most applications need

Approximate Matching

Define distance metric d on strings.

Find all text locations i

where $d(P, t_i t_{i+1} \ldots)$

is sufficiently small.

One of the most natural metrics:

HAMMING DISTANCE

Let $S_1 = s_{11}\, s_{12}\, s_{13} \cdots s_{1m}$

$S_2 = s_{21}\, s_{22}\, s_{23} \cdots s_{2m}$

$Ham(S_1, S_2)$ = The number of locations j where $s_{1j} \neq s_{2j}$

EXAMPLE: $S_1 = ABCABC$

$S_2 = ABBAAC$

(X X)

$Ham(S_1, S_2) = 2.$

String Matching With Mismatches

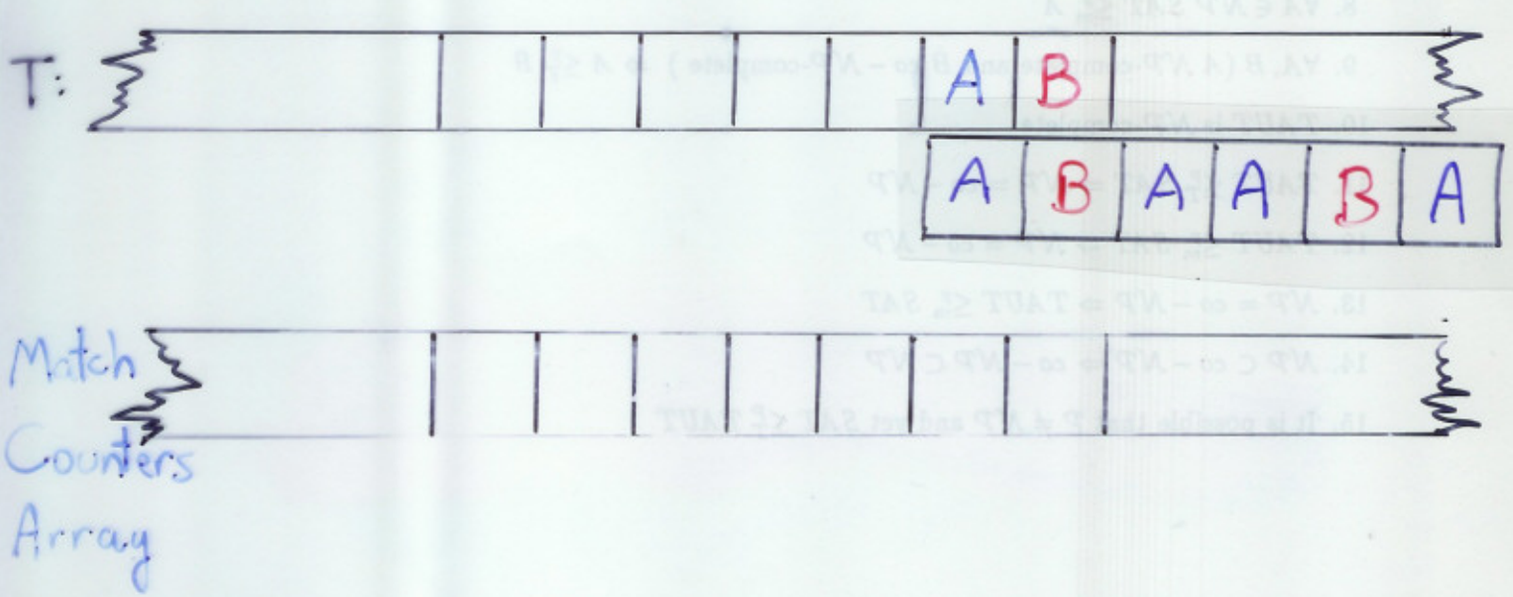
INPUT: $T = t_1 t_2 \cdots t_n$

$P = p_1 p_2 \cdots p_m$

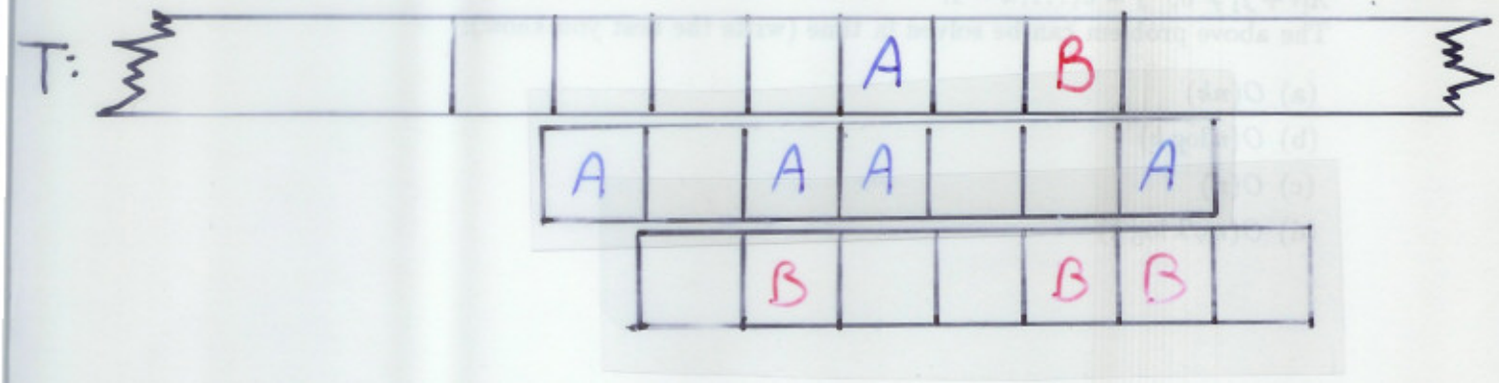
OUTPUT: $\forall\, i$ in text:

$Ham(P, t_i t_{i+1} \cdots t_{i+m-1})$

Naive Algorithm Idea: Counting (Count Matches)



We could... count matches separately for every alphabet symbol using that symbol's "mask" in pattern.



Time: $O(nm)$

Again, consider special case...

Pattern has m different symbols.

$\Rightarrow$

Every mask is of size 1

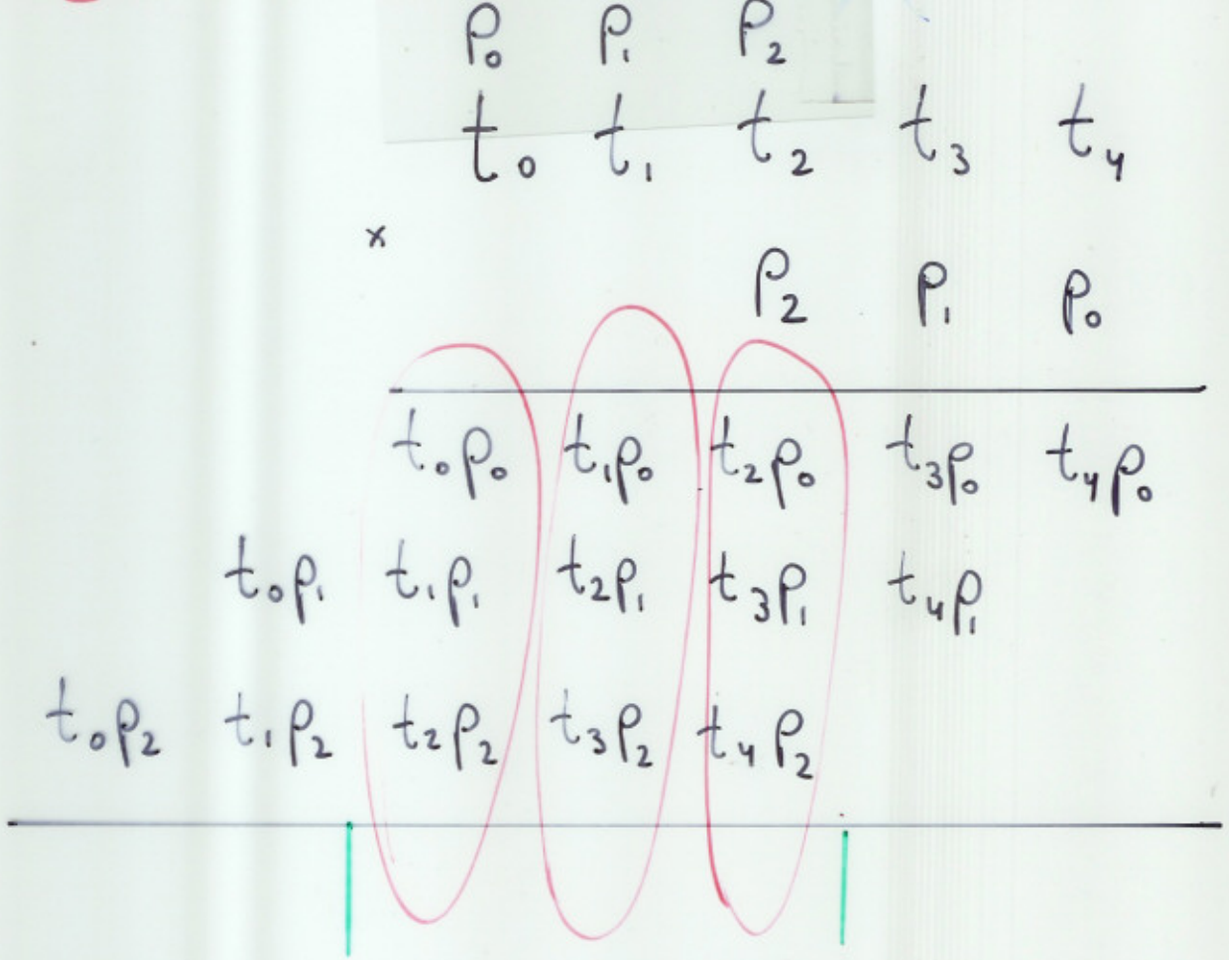
$\Rightarrow$

Time: $O(n)$

We will show: Fastest known algorithm for Pattern Matching with Mismatches Problem.

State of the Art:

1) Convolutions Method. (Fischer-Paterson '72)



Polynomial Multiplication Time: $O(n \log m)$ (using FFT).

How does this help? – Count Matches!

Define: $\chi_a(x) = \begin{cases} 0 & x \neq a \\ 1 & x = a \end{cases}$

$$\chi_a(x_1 x_2 \cdots x_m) = \chi_a(x_1) \cdots \chi_a(x_m)$$

$\chi_a(T) \times \chi_a(P^R)$

counts all matches of a

in time $O(n \log m)$.

All matches of all symbols:

$$\sum_{a \in \Sigma} \left(\chi_a(T) \times \chi_a(P^R) \right)$$

Time: $O(|\Sigma| \, n \log m)$

EXAMPLE:

Text: A B A B C A A A

Pattern: A B C A

A B C A

A B C A

A B C A

A B C A

Match Counters: 2 0 4 1 1

Multiplications:

$\chi_A(T) \times \chi_A(P^R)$

1 0 1 0 0 1 1 1

1 0 0 1

1 0 1 0 0 1 1 1

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

1 0 1 0 0 1 1 1

1 0 2 1 1

$\chi_B(T) \times \chi_B(P^R)$

0 1 0 1 0 0 0 0

0 0 1 0

0 0 0 0 0 0 0 0

0 1 0 1 0 0 0 0

1 0 1 0 0

$\chi_C(T) \times \chi_C(P^R)$

0 0 0 0 1 0 0 0

0 1 0 0

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0

0 0 1 0 0

+

2 0 4 1 1

For fixed bounded alphabets
(e.g. DNA)

we are doing great!

$O(|\Sigma| n \log m) = O(n \log m)$.

For unbounded alphabets (e.g. numbers)

we are doing awful!

$O(|\Sigma| n \log m) = O(mn \log m)$

Worse than naive ($O(mn)$).

Abrahamson's Idea (1987):
Convolutions + Divide & Conquer

Count all matches in

Time: $O(n\sqrt{m \log m})$

for all alphabets.

Galil's Question (1985):

Suppose I input text T, pattern P and number k.

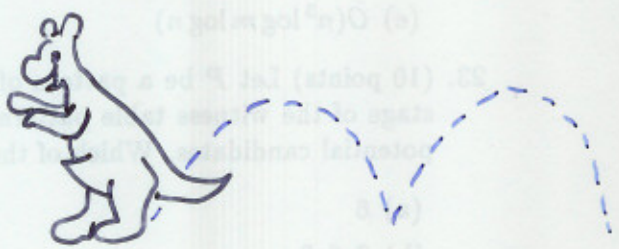
The output should be all locations i in T where

$$Ham(P, t_i t_{i+1} \cdots t_{i+m-1}) \leq k.$$

Can this be done in time $O(nk)$?

Landau & Vishkin (1986): YES!

Galil & Giancarlo (1987): Kangaroo Method.

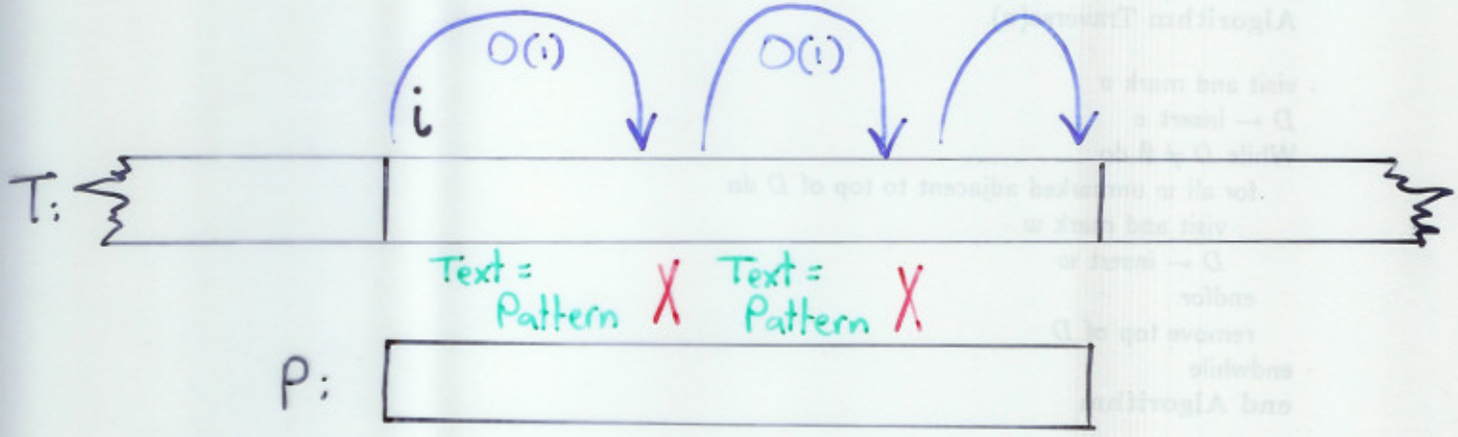


Using Suffix Trees and LCA:

- Preprocess Text & Pattern

  (in time $O((n+m)\log|\Sigma|)$)

- Subsequently, for any given text location i:



Verify in Time: $O(k)$

if $Ham(P, t_i t_{i+1} \ldots t_{i+m-1}) \leq k$.

STATE OF AFFAIRS, k-MISMATCHES PROBLEM

$O(n\sqrt{m \log m})$ Abrahamson

$O(n\sqrt{k \log k})$ New Result of this paper.

$O(nk)$ Landau-Vishkin

Galil-Giancarlo

OUR PROBLEM

INPUT: Text $T = t_1 t_2 \cdots t_n$

Pattern $P = p_1 p_2 \cdots p_m$

Max # of allowed mismatches: k

Definition: $a \in \Sigma$ is called a frequent symbol if it occurs in P at least $2\sqrt{k}$ times.

We will distinguish between two Cases:

1) At least $\sqrt{k}$ frequent symbols.

2) Less than $\sqrt{k}$ frequent symbols.

CASE 1: At least $\sqrt{k}$ frequent symbols.

Consider first $\sqrt{k}$ frequent symbols.

- For each one of them, σ, construct the mask $\chi_\sigma(P)$.

σ is frequent so the mask has more than $2\sqrt{k}$ 1's.

- Retain the leftmost $2\sqrt{k}$ 1's in the mask and change the rest to 0's.

We now have $\sqrt{k}$ masks, each with exactly $2\sqrt{k}$ 1's.

EXAMPLE: P = ABCABAACBCBDAEBACC

$k = 4$

There are $\geq \sqrt{k} = \sqrt{4} = 2$ frequent symbols

(i.e., occurring $2\sqrt{k} = 2\sqrt{4} = 4$ times at least.)

For example: A appears 6 times

B appears 5 times

C appears 5 times.

Choose A & B.

A's Mask = 1001011000000000000

B's Mask = 0100100010100000000

Counting Stage:

Run through text and count occurrences of all masks.

Time: $O(n\sqrt{k})$

Since n elements and mask size is $2\sqrt{k}$.

Important Observations:

1) Sum of all counters $\leq 2\sqrt{k}n$

2) Every counter whose value is less than k already has more than k errors!

Why? The total number of elements in all masks is $2\sqrt{k}\sqrt{k} = 2k$

The counter at every location is precisely the number of elements in the masks that match the text, starting at that location.

$\Rightarrow$

If that counter $< k$ it means already k errors!

CALCULATE:

SUM OF ALL COUNTERS: $\leq 2n\sqrt{k}$

VALUE OF POTENTIAL COUNTERS: $\geq k$

$\Downarrow$

NUMBER OF POTENTIAL COUNTERS: $\leq \frac{2n\sqrt{k}}{k} = \frac{2n}{\sqrt{k}}$.

In each potential counter:
Less than k errors in the mask elements.

But need to verify if no more than k errors altogether.

VERIFICATION: Kangaroo Method.

Time: $O(k)$ per potential location.

$\frac{2n}{\sqrt{k}}$ potential locations.

$$O\left(\frac{k\,2n}{\sqrt{k}}\right) = O\left(n\sqrt{k}\right)$$

Total.

CASE 2: x Frequent Symbols

$$x < \sqrt{k}$$

a) Count all matches of frequent symbols - one boolean multiplication per symbol.

Time: $O(x n \log m) =$

$O(\sqrt{k}\, n \log m)$

b) For non-frequent symbols, build full masks.

each one of size $< 2\sqrt{k}$

Pattern Mask Count in time

$O(n\sqrt{k})$

c) Add results of a) & b) and get total number of matches at every location.

Time:

a) $O(n\sqrt{k}\log m)$

b) $O(n\sqrt{k})$

c) $O(n)$

Total: $O(n\sqrt{k}\log m)$

BUT WE PROMISSED

$O(n\sqrt{k\log k})$

!?!

FINE TUNING

1) We also have a

$$O\left(n + \left(\frac{nk^3}{m}\right)\log m\right)$$ time

algorithm for the k-mismatch problem.

For $k < m^{\frac{1}{6}} = \frac{m^{\frac{1}{3}}}{m^{\frac{1}{6}}} = \left(\frac{m}{\sqrt{m}}\right)^{\frac{1}{3}} < \left(\frac{m}{\log m}\right)^{\frac{1}{3}}$

The above algorithm runs in linear time.

So we only run our algorithm for $k > m^{\frac{1}{6}}$.

But for $k > m^{\frac{1}{6}}$,

$$\log m = O(\log k^6) =$$

$$= O(\log k)$$

Thus algorithm's time is:

$$O(n\sqrt{k}\log m) =$$

$$O(n\sqrt{k}\log k).$$

2) The time was distributed:

$O(n\sqrt{k}\log k)$

for frequent symbols

$O(n\sqrt{k})$

for non-frequent symbol.

Tradeoff: Define frequent as having $2\sqrt{k\log k}$ elements.

Then need only $\sqrt{\frac{k}{\log k}}$

frequent elements for case 1).