University of Chile
Department of Computer Science
Gonzalo Navarro

University of Joensuu
Department of Computer Science
Kjell Fredriksson

Average-Optimal Multiple Approximate String Matching
\[\Omega(n + \log^2 m) / m \text{ for } r \text{ patterns (this work).}\]

- On average at least \(\Omega(n + \log m) / m\) character comparisons for an approximate matching of one pattern (Chang & Marri, 94).

- Applications: virus and intrusion detection, spelling, text retrieval (synonyms and thesaurus) computational biology, batched searching.

- Given text \(T[1..n]\) and pattern \(P[1..m]\) over some finite alphabet \(\Sigma\) of size \(|\Sigma| > 0\),

The classic approximate string matching problem.

\(\Omega\)
– Other less interesting ones.

– Error levels \( k/w > 1/\log^2(m) \). Best in practice.

– Works for low and medium \( O(\log^2(m/n)m) \). Works but can be improved to \( O(\log^2(m/n)m) \) pre-processing.

– Partitioning into exact search: \( O(\text{average}) \) on average \( O(n) \).

–


\[ \text{Previous Work} \]
Displacing partitioning algorithm to intermediate error levels.

In practice: best algorithm for low error levels, few or many patterns,

- Aligned in important gap.

In theory: best algorithm for low and medium error levels, few or many

Preprocessing time is \( \mathcal{O}(m) \) and space is \( \mathcal{O}(\sqrt{m}) \), and space is very efficient in practice too.

Several improvements to the basic algorithms very efficient in practice too.

Another average time algorithm for \( \mathcal{O}(u) \) \n
\[
\frac{\sigma^2}{1} - \frac{2}{1} > \frac{w}{h} \mathcal{O}
\]

This holds for \( \mathcal{O}(u) \) \n
Our algorithm runs in \( \mathcal{O}(w/(\omega w + \log \gamma n)) \) \n
We show that the lower bound for the problem is \( \mathcal{O}(n) \) \n
Multiple patterns.

We have generalized the (optimal) algorithm of Chaniak & Marr (1994) to

Main Results
(Chang & Marriott, 94).

Summing over these two basic facts gives \( \Omega(n(m/((m(\log^0 m) + \log m))) \) lower bound \( \Omega(n/m) \).

Since + 1 characters has to be examined to skip the block, this leads to lower bound \( \Omega(n/m) \). (Chang & Marriott, 94) generalized this for approximate matching: At least

At least \( \Omega(n \log^0 m (m/\log^0 m)) \) comparisons to solve the exact string matching problem.

At least \( \Omega(n \log^0 m (m/\log^0 m)) \) comparisons are needed on average.

In order to discard all these positions, at least \( \Omega(n \log^0 m) \) characters.

There are \( m \) possible positions for the pattern to occur.

\( \leq \)

I. (Xiao, 79): The text is divided into blocks of length \( 2m \), and

The lower bound for approximate string matching for one pattern:

The lower bound
wasn't proved before.

Interestingly, there exists several optimal exact multi-pattern search algorithms (e.g. MULTIBDM (Crochemore & Rytter, 94)), but the optimality
approximate searching:

Summing again leads to \( \Omega \left( \log \frac{\mathsf{m}}{\mathsf{n}} \right) \) lower bound for exact searching.

This needs at least \( \Omega \left( \log \frac{\mathsf{m}}{\mathsf{n}} \right) \) character inspections on average.

\( \Rightarrow \)

We need to discard. We again divide the text in blocks of length \( 2m - 1 \), but now there are \( T_m \)

The idea is the same as in the Yao's proof.

We adapt this to multi-pattern matching.
The Basic Algorithm: Preprocessing

1. Build a table $D$ as follows:

2. For every string $S$ of length $|D|$ (a-gram), search for $S$ in $A$.

3. Store in $D$ the smallest number of differences needed to match $S$.

$D$ (a number between 0 and $\lambda$).

$O$ requires space for $\lambda$ entries and can be computed in $O(\lambda m)$ time.

$D$
We want to find this block. E.g. For \( m = 14 \) and \( k = 2 \), we have \( q = 6 \).

\[ \lceil \frac{z/(k-m)}{2} \rceil = q \]

\( \text{Observation: any approximate occurrence of } \pi, \text{ which has length of at least } m - k, \text{ contains at least one whole block.} \)

- Divide the text into non-overlapping blocks of length

**The Basic Algorithm: Filtering**
The Basic Algorithm: Filtering

1. Take the first $\mathcal{F}$-gram of the block, $S^1$, and obtain

2. Take the next $\mathcal{F}$-gram, $S^2$, and obtain

3. If, before reaching the end of the block, the sum $\frac{q}{y} \cdot \frac{p}{x} < \frac{f}{x}$, the block can be skipped.

4. If at the end of the block, the sum $\frac{q}{y} = \frac{f}{x}$, the block must be verified.
Skip the block

\[ I = [s]a \]
\[ \gamma = [s]a \]

\[ \gamma < \gamma \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]

\[ \gamma = [s]a \]
bit-parallelism (Mayers, 99).

This requires worst case time \( O(m^2) \) for each pattern.

\[ \Delta P \in \Phi \]

\[
\text{For each block } T_1 \ldots T_q \text{ that has to be verified run the classical}
\]

\[ \text{The Basic Algorithm: Verification} \]
3. If the current node is leaf, then verify with dynamic programming.

children.

2. If the either triggers "verification", recursively filter with for the $A$ (Table for the current node).

1. Filter the current block with the table for the current node.

- Filter recursively (starting from the root node):
  - Build the table for each node, taking only the corresponding patterns into account.
  - Only one pattern in each leaf:
    - Each child node contains only the half of the patterns of its parent.
    - The root node contains all the patterns.

- Build a binary tree for the patterns.

Extensions: (Binary) pattern hierarchy
But it needs more space.

In practice much faster except for very small k/m.
This increases tree arity, reduces verification time and space.

short.

We manage several counters in a single integer and update them all in one

\[
[S] D^{\frac{\gamma}{1 \leq \epsilon}} \leq \gamma \log_2 \left( \frac{1}{1 + \gamma + \epsilon} \right) \leq B
\]

If we allow at most \( \gamma \) errors, then \( B \) is

Bit-parallel counters \( \gamma \) higher arity pattern hierarchy.
The space requirement is \( O^{\rho/2}w = o \), where \( \rho \) is a constant. If we take \( m/w \) as a constant,
\[ p/(\rho\log \log m + 2 \log \log \log m) = o \]

The preprocessing cost is \( (m/(\rho\log m)) = o \), which is optimal for the 
\( (\rho^\wedge)/(\rho) = o - \rho^\wedge > m/w \)

This holds for the parallel and verification only.

Analysis
The result is an algorithm that takes \( O(n \log n) \) time for \( k/m > \frac{1}{2} - \frac{1}{16} \).

move to the next block.

If the \( t \)-grams inside the window add up more than \( k \) differences, we can

\[ \frac{|G + G| \ldots |G + G + 1 + G|}{G} = \frac{1}{t - 1} \quad \frac{t}{(t)} \frac{O}{1} \]

Consider text blocks for the form \( t \) [\( \leq |G + G + 1 + G| \) where \( t \leq t \)-grams.

Blocks overlap with each other by \( t - 1 \)-grams.

A sliding window of \( t \)-grams, where

\( O(\log^2 \log n) \leq \frac{1}{2} - \frac{1}{2} \lfloor \log n \rfloor \)

\( O^{\log^2 \log n} \) is partly due to the use of fixed text blocks of length \( m - k \) for \( \frac{1}{2}/\frac{1}{3} \).

The algorithm cannot cope with difference ratios \( k/m \) beyond \( 1/3 \).

Linear time algorithm for higher error levels
which allowed \( O(1) \) time access to the \( k \)-grams.

The texts were stored using only 2 (DNA) and 5 bits (protein) per character,

- The patterns were 64 characters long.
- The text was for protein and ASCII.
- 64MB characters for DNA,
- 16MB characters for protein and ASCII.
- The test data for DNA and protein alphabets was randomly generated.

Experiments for alphabet sizes \( \sigma = 20 \) (protein) and \( \sigma = 256 \) (DNA).

In 2GHz Pentium 4, with 512MB RAM, with Linux 2.4.

Implementation in C, compiled using GCC 3.2.1 with full optimizations, run

Experimental Results
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>60.09</th>
<th>0.04</th>
<th>( t = 4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.55</td>
<td>0.40</td>
<td>60.00</td>
<td>0.01</td>
<td>( t = 3 )</td>
</tr>
<tr>
<td>0.07</td>
<td>0.03</td>
<td>10.00</td>
<td>0.00</td>
<td>( t = 2 )</td>
</tr>
<tr>
<td>0.02</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>( t = 1 )</td>
</tr>
<tr>
<td>( \tau = 1024 )</td>
<td>( \tau = 256 )</td>
<td>( \tau = 64 )</td>
<td>( \tau = 1 )</td>
<td>Protein</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>0.24</th>
<th>0.08</th>
<th>( t = 10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.19</td>
<td>12.00</td>
<td>3.02</td>
<td>0.38</td>
<td>( t = 8 )</td>
</tr>
<tr>
<td>1.17</td>
<td>0.58</td>
<td>0.15</td>
<td>0.02</td>
<td>( t = 6 )</td>
</tr>
<tr>
<td>0.08</td>
<td>0.04</td>
<td>0.01</td>
<td>0.00</td>
<td>( t = 4 )</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>DNA</td>
</tr>
</tbody>
</table>

Preprocessing times in seconds:
We cannot use the optimal $\gamma$ values: 18 (DNA), 10 (proteins), and 6 (ASCII).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>$4.26$</th>
<th>$3 = \gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1.06$</td>
<td>$2.60$</td>
<td>$0.59$</td>
<td>$0.01$</td>
<td>$2 = \gamma$</td>
</tr>
<tr>
<td>$0.06$</td>
<td>$0.01$</td>
<td>$0.01$</td>
<td>$0.00$</td>
<td>$1 = \gamma$</td>
</tr>
<tr>
<td>$r = 1024$</td>
<td>$r = 256$</td>
<td>$r = 64$</td>
<td>$r = 1$</td>
<td>ASCII</td>
</tr>
</tbody>
</table>
Search times (s) for DNA, for EP and for $f$, with $l = 8$, and $j = 64$. 

Running times for varying $k$.
Running times for varying $k$.

Times for protein for EP and for $r = 3$, with $r = 64$, $r = 256$, and $r = 1024$. 
Times for ASCII, for EP and for $\mathcal{F}$ with $\mathcal{F} = 3$ with $r = 64$, $r = 256$, and $r = 1024$. 
Conclusions...
... and future work...