String Algorithms and Data Structures
Approximate Pattern Matching

CS 199-225
Brad Solomon

November 7, 2022
Learning Objectives

Review exact pattern matching and introduce approximate matching

Formally define a mismatch vs an edit

Discuss strategies for efficient approximate pattern matching…

… With mismatches

… With edits
# Suffix-Based Index Review

<table>
<thead>
<tr>
<th></th>
<th>Suffix tree</th>
<th>Suffix array</th>
<th>FM Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time: Does P occur?</strong></td>
<td>$O(n)$</td>
<td>$O(n \log m)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>Time: Count $k$ occurrences of P</strong></td>
<td>$O(n + k)$</td>
<td>$O(n \log m)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>Time: Report $k$ locations of P</strong></td>
<td>$O(n + k)$</td>
<td>$O(n \log m + k)$</td>
<td>$O(n + k)$</td>
</tr>
<tr>
<td><strong>Space</strong></td>
<td>$O(m)$</td>
<td>$O(m)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td><strong>Needs T?</strong></td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td><strong>Bytes per input character</strong></td>
<td>$&gt;15$</td>
<td>$\approx 4$</td>
<td>$\approx 0.5$</td>
</tr>
</tbody>
</table>

$m = |T|$, $n = |P|$, $k = \#$ occurrences of $P$ in $T$
Limitations of exact pattern matching

But what if I was actually trying to look up ‘string’?
Limitations of exact pattern matching

If I ban “bad word”, what happens to “b@d w0rd”?
Approximate Pattern Matching

Scholarly articles for de Bruijn graph
- IDBA—a practical iterative de Bruijn graph de novo - Peng - Cited by 269
- ... metagenomics assembly via succinct de Bruijn graph - LI - Cited by 1244
- Space-efficient and exact de Bruijn graph ... - Chikhi - Cited by 245

De Bruijn graph

In graph theory, an n-dimensional De Bruijn graph of m symbols is a directed graph representing overlaps between sequences of symbols. It has m^n vertices, consisting of all possible length-n sequences of the given symbols; the same symbol may appear multiple times in a sequence. Wikipedia
Approximate Pattern Matching

Score = 248 bits (129), Expect = 1e-63
Identities = 213/263 (80%), Gaps = 34/263 (12%)
Strand = Plus / Plus

Query: 161 atatcaccacgtaaagggtgactccaactcca---ccactccattttgtcagataatgc 217
Sbjct: 481 atatcaccacgtaaagggtgactccaacttattgatagtgttttatgtcagataatgc 539

Query: 218 ccgatgatcatgtcagctccaccgattttgtgagaacagcactccgtttccagc 277
Sbjct: 540 ccgatgactttgtcatgcagctccaccgattttg--ttccgtttccagc 586

Query: 278 c-gtgcc--aggtgctgcctcagattcaggtatatgcgcgtcgatgatgatgc 334
Sbjct: 587 caatgacgta-gtgctgcctcagattcaggtatatgcgcgtcgatgatgatgc 645

Query: 335 ttgctgattacgtgcagctttcccttcaggcgga----------ccagcgcattcgcttcagc 382
Sbjct: 646 ttgctgattacgtgcagctttcccttcaggcgga----------ccagcgcattcgcttcagc 705

Query: 383 ctccatatc-accacgtcaaagg 404
Sbjct: 706 atccatatcaaccacgtcaaagg 728
Approximate Pattern Matching

Input: A text $T$, a pattern $P$, and a distance $d$

Output: All positions in $T$ where $P$ has at most $d$ mismatches or edits

$P$: word
$T$: There would have been a time for such a word

Alignment 1: word
Alignment 2: word

Not a match!
Distance 2 match!

Match!
Distance 0 match!
Approximate Pattern Matching

What is the distance between these two strings?

\[ X: 10011 \]
\[ Y: 00110 \]

Hamming distance is 3!

Edit distance is 2!
Approximate Pattern Matching

A **substitution** replaces one character with another

Described as the character swap needed to *convert* $T$ to $P$


$P$: G T A A C G G C G

Mismatch
(Substitution)
Hamming Distance

The minimum number of substitutions to turn one string into another.

\[X: \textcolor{red}{G} \textcolor{green}{A} \textcolor{red}{G} \textcolor{green}{G} \textcolor{red}{T} \textcolor{green}{A} \textcolor{red}{G} \textcolor{green}{C} \textcolor{red}{G} \textcolor{green}{G} \textcolor{red}{C} \textcolor{green}{G} \textcolor{red}{T} \textcolor{green}{T} \]

\[Y: \textcolor{green}{G} \textcolor{red}{T} \textcolor{green}{G} \textcolor{red}{G} \textcolor{green}{T} \textcolor{red}{A} \textcolor{red}{A} \textcolor{green}{C} \textcolor{green}{G} \textcolor{green}{G} \textcolor{green}{G} \textcolor{green}{G} \textcolor{green}{G} \textcolor{green}{T} \textcolor{green}{T} \]

Hamming distance = 3

\[X: \textcolor{red}{T} \textcolor{red}{G} \textcolor{green}{G} \textcolor{green}{C} \textcolor{green}{C} \textcolor{green}{G} \textcolor{green}{C} \textcolor{green}{G} \textcolor{green}{C} \textcolor{green}{A} \textcolor{red}{A} \textcolor{red}{A} \textcolor{red}{A} \textcolor{red}{A} \textcolor{red}{A} \textcolor{red}{C} \textcolor{red}{A} \textcolor{red}{G} \textcolor{red}{C} \]

\[Y: \textcolor{red}{T} \textcolor{green}{G} \textcolor{green}{A} \textcolor{green}{C} \textcolor{red}{C} \textcolor{green}{G} \textcolor{green}{C} \textcolor{green}{G} \textcolor{green}{C} \textcolor{red}{A} \textcolor{red}{A} \textcolor{red}{A} \textcolor{red}{A} \textcolor{red}{A} \textcolor{red}{C} \textcolor{red}{A} \textcolor{red}{G} \textcolor{red}{C} \textcolor{red}{T} \]

Hamming distance = 6
Hamming Distance

The minimum number of substitutions to turn one string into another.

\[ \text{Hamming distance} = 5 \]

\[ \text{Hamming distance} = 6 \]
Approximate Pattern Matching

An **insertion** adds a character, shifting all other characters back.

Insertion is relative! What edits *convert* $T$ to $P$?

$T$: `GGAAAAAGAGGTAGC - GCGTTTAACAGTAG`

$P$: `GTAGCGGCGG`

Insertion
Approximate Pattern Matching

An **deletion** removes a character, shifting all other characters forward.

Deletion is relative! What edits *convert* $T$ to $P$?

- **$T$:** GGAAAAAAGAGGTAGCGGGC GTTTTAACAGTAG
- **$P$:** GT - GC GG GC G

Deletion
Edit Distance

The minimum number of substitutions, insertions, or deletions (*edits!*) needed to turn one string into another (from X to Y)!

<table>
<thead>
<tr>
<th>X:</th>
<th>T G G C C G C G C A A A A A C A G C -</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y:</td>
<td>T G A C C G C G C A A A A - C A G C T</td>
</tr>
</tbody>
</table>

*Edit distance = 3*

<table>
<thead>
<tr>
<th>X:</th>
<th>G C G C T</th>
<th>G C G C T</th>
<th>G G C C T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y:</td>
<td>- - G C T</td>
<td>G C - - T</td>
<td>G - - C T</td>
</tr>
</tbody>
</table>

*Edit distance = 2*
Edit Distance

The minimum number of substitutions, insertions, or deletions (edits!) needed to turn one string into another (from X to Y)!

$X$: \[ G \ G \ C \ C \ G \ G \ C \]

$Y$: \[ C \ C \ G \ G \ G \ G \ G \]

$Edit\ distance = 5$

$X$: \[ T \ A \ T \ A \ T \ A \ - \]

$Y$: \[ - \ - \ A \ T \ A \ T \ A \ T \]

$Edit\ distance = 2$
Edit Distance

```
cart  <->  cart
       |  1 deletion  |
|       |             |
carry  <->  carrot
       |  1 insertion  |
|       |             |
caret
```

- **1 substitution**
- **1 deletion**
- **1 insertion**
Edit Distance

Score = 248 bits (129), Expect = 1e-63
Identities = 213/263 (80%), Gaps = 34/263 (12%)
Strand = Plus / Plus

Query: 161 atatcaccacgtcaaagtgactccaactcca---ccactccatattttttgttcagataatgc 217
Sbjct: 481 atatcaccacgtcaaagtgactccaact-tattgatagtg tgtttatgttcagataatgc 539

Query: 218 ccgatgatcatgtcatgctagctccaccgattttgagaaacgacagcgcacctccggtccagc 277
Sbjct: 540 ccgatgacttttgtcatgctagctccaccgattttg-g--------------ttccgtccagc 586

Query: 278 c-gtgcc--aggtgcctgcctcagatttcatgcggctcagctcatattgcgtatatgcg 334
Sbjct: 587 caatgacgta-gtgcgtgcctcagattcaggttatgctcggctcaattgcgtatatgcg 645

Query: 335 ttgctgattacgtgcagctttcccttcaggcggga---------------ccagcctattcgc 382
Sbjct: 646 ttgctgattacgtgcagctttcccttcaggcggga---------ttcatacagcggccagccatcgc 705

Query: 383 ctccatatc-accacgtcaagg 404
Sbjct: 706 atccatatcaaccacgtcaagg 728
Approximate Pattern Matching

**Input:** A text $T$, a pattern $P$, and a distance $d$

**Output:** All positions in $T$ where $P$ has at most $d$ mismatches or edits

**Hamming Distance:** Min number substitutions (mismatches)

**Edit Distance:** Min number edits (substitution, insertions, deletions)
Approximate Pattern Matching

\[ \Sigma = \{0, 1\} \quad P = 000 \]

Hamming Distance 1 strings:

Edit Distance 1 strings:
Approximate Pattern Matching

$\Sigma = 0, 1 \quad P = 000$

Hamming Distance 1 strings: 100 010 001

Edit Distance 1 strings: 100 1000 00
010 0100
001 0010
000 0000
Approximate Pattern Matching

\[ P = abb \quad d = 1 \]

Using \textit{Hamming} distance, what are valid approximate matches for \( P \)?

Using \textit{edit} distance, what are valid approximate matches for \( P \)?

A) aba \hspace{2cm} B) aabb

C) bbb \hspace{2cm} D) ab
Approximate Pattern Matching

How do we find all approximate matches for a pattern in a text?
Approximate Pattern Matching

How do we find all approximate matches for a pattern in a text?

\( P: \text{ word} \)

\( T: \) There would have been a time for such a word

word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
word word word word word word word word word word
Approximate Pattern Matching

How do we find all approximate matches for a pattern in a text?

Can we use our efficient exact pattern matching algorithms?
Approximate Pattern Matching

Can we use our efficient exact pattern matching algorithms?

For Hamming distance (mismatches), we can!

\[ P \]

101110101101101010001010
Approximate Pattern Matching

Can we use our efficient exact pattern matching algorithms?

For Hamming distance (mismatches), we can!

\[ P \]

10111010111011010001010
Approximate Pattern Matching

Can we use our efficient exact pattern matching algorithms?

For Hamming distance (mismatches), we can!
Approximate Pattern Matching

Can we use our efficient exact pattern matching algorithms?

For Hamming distance (mismatches), we can!

If $P$ occurs in $T$ with 1 mismatch, then $u$ or $v$ has no mismatch

We can search for $u$ and $v$ in $T$ as a proxy for $P$!
Approximate Pattern Matching

Can we use our efficient exact pattern matching algorithms?

\[ P: \text{word} \]

\[ T: \text{There would have been a time for such a word} \]

\[ u: \text{wo \hspace{1cm} wo} \]

\[ v: \text{rd \hspace{1cm} rd} \]

1) Search for the two half-patterns of \( P \)

2) Compute the number of mismatches for each half-match!

3) Return all matches (but don’t duplicate!)
Approximate Pattern Matching

If $P$ occurs in $T$ with up to $k$ mismatches…
Approximate Pattern Matching

If $P$ occurs in $T$ with up to $k$ mismatches, then if we split $P$ into $k+1$ partitions, at least one of $p_0, p_1, ..., p_k$ must appear with 0 mismatches.
Approximate Pattern Matching

If $P$ occurs in $T$ with up to $k$ mismatches, then if we split $P$ into $k+1$ partitions, at least one of $p_0, p_1, ..., p_k$ must appear with 0 mismatches.

<table>
<thead>
<tr>
<th>XXX</th>
<th></th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XX</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

5 partitions
4 mismatches ($X$)
Approximate Pattern Matching

1) **Pigeonhole principle:** we will have at least one exact match!

\[ k+1 \text{ pigeons, } k \text{ holes?} \]

At least one hole has two pigeons!

\[ k \text{ pigeons, } k+1 \text{ holes?} \]

At least one hole is empty!
Approximate Pattern Matching

2) Verifying a match is $O(P)$
Seed and Extend

Using an exact-match ‘seed’ and ‘extending’ the match is efficient!
Seed and Extend

Exact match of *seeds* reduce the search space

Let $T$: There would have been a time for such a word word word

$p$: word

Only consider mismatches while verifying a seed hit
Seed and Extend

Exact match of **seeds** reduce the search space

\[ P: \text{word} \]

\[ T: \text{There would have been a time for such a word word word} \]

Consider the likelihood of seeing ‘wo’ or ‘rd’ by chance (assuming random string):

\[
256 \text{ characters : } \frac{1^2}{256} = 0.000015
\]
Approximate Pattern Matching in Genomics

**Partition Seed:** Length ~40
CTCAAAACTCCTGACCTTTGGGTGATCCACCCGCCTAGGCCTTC

**T:** Length 3 billion

\[ \text{Likelihood of random seed string:} \]
\[ \frac{1}{4^{40}} = 8.27 \times 10^{-25} \]

# of times seed will occur by chance in T:

\[ 2.48 \times 10^{-15} \]

Likelihood * (~ length)
Approximate Pattern Matching

“Seed and extend” approach to pattern matching

\[
\begin{array}{cccccc}
\cdots & p_0 & p_1 & p_2 & p_3 & \cdots & p_k \\
\end{array}
\]

Advantages

- Reuse favorite exact matching algs; fast and easy
- Flexible; works for Hamming and edit distance*

Disadvantages

- Large \( k \) yields small partitions matching many times by chance; lots of verification work
- \( k+1 \) exact matching problems, one per partition

* we don’t know how to do edit distance verification yet
Approximate Pattern Matching

If $P$ occurs in $T$ with up to $k$ mismatches, then if we split $P$ into $k+1$ partitions, at least one of $p_0, p_1, \ldots, p_k$ must appear with 0 mismatches.

$P = \text{AAAAA} \quad d = 3$

How many partitions?

What is the characters in each partition(s)?

$T = \text{BBBBBBBAABBBBBB} \ldots$
Approximate Pattern Matching

If $P$ occurs in $T$ with up to $k$ mismatches, then if we split $P$ into $k+1$ partitions, at least one of $p_0, p_1, \ldots, p_k$ must appear with 0 mismatches.

\[ P = \text{B A A A A A A} \quad d = 1 \]

\[ T = \ldots \text{B B B B B B A A A A A B A A} \ldots \]
Assignment 10: a_pigeon

Learning Objective:

Preprocess text into kmers and a hash table

Use pigeonhole principle to perform approximate matching

Consider: Do the partitions need to be contiguous runs of characters? Do they need to all be the same length?
vector<int> approximate_search(fname, P, mm)

Input:

- **string fname**: The file storing the text T
- **string P**: The pattern text
- **int mm**: The number of allowed mismatches

Output:

- **vector<int>**: The index positions in T of all approximate matches

```
\begin{array}{cccc}
p_0 & p_1 & p_2 & p_3 & p_4 \\
\text{verify} & \text{verify} & \text{verify} & \text{match} & \text{verify} \\
\end{array}
\end{array}
```

```
T
```
vector<Seed> partitionPattern(string P, int np)

Input:

string P: The input pattern — can be large this week!

int np: The number of non-overlapping partitions to split P

Output:

vector<Seed>: Vector of partitioned strings and their index

typedef std::pair<std::string, int> Seed;

P: ABCDEFGH          np = 2          { {ABCD, 0}, {EFGH, 4} }
kmerMap text_to_kmer_map(string & T, int k)

Input:

string & T: The input text — can be very large this week!

int k: The fixed size for each kmer (substring)

Output:

kmerMap: unordered_map<string, vector<int>>

T: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Bonus Slides
**FM Index w/ mismatches**

Start with shortest suffix, then match successively longer suffixes

Keep track of mismatches for each suffix

$P = \text{aba}$

<table>
<thead>
<tr>
<th>$F$</th>
<th>$L$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$$ a b a a b a_0$</td>
<td></td>
</tr>
<tr>
<td>$a_0$ $$$ a b a a b</td>
<td></td>
</tr>
<tr>
<td>$a_0$ a b a $$$ a b</td>
<td></td>
</tr>
<tr>
<td>$a_1$ b a $$$ a b a_1</td>
<td></td>
</tr>
<tr>
<td>$a_2$ b a a b a $$$</td>
<td></td>
</tr>
<tr>
<td>$a_3$ b a a b a $$$</td>
<td></td>
</tr>
<tr>
<td>b a $$$ a b a a_2</td>
<td></td>
</tr>
<tr>
<td>b a a b a $$$ a_3</td>
<td></td>
</tr>
</tbody>
</table>

Easy to find all the rows beginning with a
FM Index w/ mismatches

We have rows beginning with \( a \), now we want rows beginning with \( ba \)

\[
P = aba
\]

\[
\begin{array}{c|c}
F & L \\
\hline
$ & a b a a b a_0 \\
a_0 & $ a b a a b \\
a_1 & a b a a $ a b \\
a_2 & b a a a b a_1 \\
a_3 & b a a b a $ \\
b & a b a $ a b a_2 \\
b & a a b a b a_3 \\
\end{array}
\]

\[
P = aba
\]

\[
\begin{array}{c|c}
F & L \\
\hline
$ & a b a a b a_0 \\
a_0 & $ a b a a b \\
a_1 & a b a a $ a b \\
a_2 & b a a a b a_1 \\
a_3 & b a a b a $ \\
b & a b a $ a b a_2 \\
b & a a b a b a_3 \\
\end{array}
\]

No longer have just one search range!
FM Index w/ mismatches

We have rows beginning with $ba$, now we seek rows beginning with $aba$

$P = aba$

We have rows beginning with $ba$, now we seek rows beginning with $aba$

$P = aba$

Only works for Hamming Distance (mismatches)!