String Algorithms and Data Structures
Approximate Pattern Matching

CS 199-225
Brad Solomon

April 10, 2023
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>Time: Does P occur?</td>
<td>$O(n)$</td>
<td>$O(n \log m)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Time: Count $k$</td>
<td>$O(n + k)$</td>
<td>$O(n \log m)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>occurrences of P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time: Report $k$</td>
<td>$O(n + k)$</td>
<td>$O(n \log m + k)$</td>
<td>$O(n + k)$</td>
</tr>
<tr>
<td>locations of P</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Space</td>
<td>$O(m)$</td>
<td>$O(m)$</td>
<td>$O(m)$</td>
</tr>
<tr>
<td>Needs T?</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Bytes per input</td>
<td>&gt;15</td>
<td>~4</td>
<td>~0.5</td>
</tr>
<tr>
<td>character</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$m = |T|$, $n = |P|$, $k = \# \text{ occurrences of } P \text{ in } T$
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

Properties · Dynamical systems
Approximate Pattern Matching

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

Query: 161 atatcaccacgtcaaaggtgactccaaactcca---ccactccatttttgtagataatgc 217
Sbjct: 481 atatcaccacgtcaaaggtgactccaaact-tattgatagtttttatgtgtagataatgc 539

Query: 218 ccgatgatcatgtcatgcagctccaccgattgtgagaacgacagcgactttccggtccagc 277
Sbjct: 540 ccgatgacttttgcagctccaccgattttg-------------ttccgtccagc 586

Query: 278 c-gtgcc--aggtgctgcctcagattcaggttatgccgctcaattcgctgcgtatatcgc 334
Sbjct: 587 caatgacgta-gtgctgcctcagattcaggttatgcggctcaattcgctgggtatatcgc 645

Query: 335 ttgctgattacgtgcagctttcccttcaggccggga------------ccagccatcctgct 382
Sbjct: 646 ttgctgattacgtgcagctttcccttcaggccgggattcatacagcggccagccatcctgct 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?

\[ X: 10011 \]

\[ Y: 00110 \]
Approximate Pattern Matching

What is the distance between these two strings?

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

**Hamming distance is 3!**

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

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

**Edit distance is 2!**
Approximate Pattern Matching

How can I describe the relationship between two strings?

\[ \begin{align*}
X: & \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \\
| & | \\
Y: & \ 0 \ 0 \ 1 \ 1 \ 0
\end{align*} \quad \begin{align*}
X: & \ 1 \ 0 \ 0 \ 1 \ 1 \ - \\
| & | | | \\
Y: & \ - \ 0 \ 0 \ 1 \ 1 \ 0
\end{align*} \]
Approximate Pattern Matching

A **substitution** replaces one character with another. Described as the character swap needed to *convert* $T$ to $P$.

$T$: GGAAAAAGAGGTAGCGGCGTTTAACAGTAG

$P$: GTAACA CGGCG

Mismatch
(Substitution)
Hamming Distance

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

\[ X: \text{GAGGTAGCGGCGTT} \]
\[ Y: \text{GTGGTAACGGGGTT} \]
\[ \text{Hamming distance} = 3 \]

\[ X: \text{TGGCCGCGCAAAAACAGC} \]
\[ Y: \text{TGACCGCGCAAAACAGCT} \]
\[ \text{Hamming distance} = 6 \]

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

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

\[
\begin{align*}
X: & \quad \text{GGCCGGC} \\
Y: & \quad \text{CCGGGGG} \\
\mid & \quad \mid \\
\text{Hamming distance} & = 5
\end{align*}
\]

\[
\begin{align*}
X: & \quad \text{TATATA} \\
Y: & \quad \text{ATATAT} \\
\text{Hamming distance} & = 6
\end{align*}
\]
Approximate Pattern Matching

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

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

$T$: G G A A A A A A A G A G G T A G C - G C G T T T A A C A G T A G

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

Insertion
Approximate Pattern Matching

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

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

$T: \text{GGAAAAAAGAGGTAGCGGGCGTTTAACAGTAG}$

$P: \text{GT-GCGGGCG}$
The minimum number of substitutions, insertions, or deletions (edits!) needed to turn one string into another (from X to Y)!

\[ \begin{align*} 
X: & \quad T G G C C G C G C A A A A A C A G C - \\
& \quad \mid \mid \mid \mid \mid \mid \mid \mid \mid \mid \mid \mid \mid \mid \\
Y: & \quad T G A C C G C G C A A A A - C A G C T \\
\end{align*} \]

\[ \text{Edit distance} = 3 \]

\[ \begin{align*} 
X: & \quad G C G C T \\
& \quad \mid \mid \mid \\
Y: & \quad - - G C T \\
\end{align*} \]

\[ \begin{align*} 
X: & \quad G C G C T \\
& \quad \mid \mid \mid \\
Y: & \quad G C - - T \\
\end{align*} \]

\[ \begin{align*} 
X: & \quad G G C C T \\
& \quad \mid \mid \mid \\
Y: & \quad G - - C T \\
\end{align*} \]

\[ \text{Edit distance} = 2 \]
Edit Distance

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

\[
X: \quad \text{GGCCGGC} \quad \text{GGCCGGC} - - \\
\quad | \quad | \quad | \quad | \quad | \quad | \\
Y: \quad \text{CCGGGGG} \quad - - \text{CCCCGGGGG} \\
\]

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

\[
X: \quad \text{TATATA} \quad - \\
\quad | \quad | \quad | \quad | \quad | \quad | \\
Y: \quad - \text{ATATATA} \\
\]

\[\text{Edit distance} = 2\]
Edit Distance

1 deletion

cart ← carrt

1 substitution

carry

caret

1 insertion

carrot
**Edit Distance**

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

**Query:**
161 atatcaccagtc...aagg 217
218 c...gtgcc--aggtg... 277
278 c-g...tgctgattac... 334
335 ttgctgattac... 382
383 ctccatatc-accacgt... 404

**Sbjct:**
481 atatcaccagtc...aagg 539
540 ccgatgac...tttetgtc... 586
587 caatgacgta-gtgctg... 645
646 ttgctgattac... 705
706 atccatatcaaccag... 728

The alignment shows:
- **Substitution**
- **Deletion**
- **Insertion**
Approximate Pattern Matching

How can I describe the relationship between two strings?

\[ X: 1\ 0\ 0\ 1\ 1 \quad Y: 0\ 0\ 1\ 1\ 0 \]

**Edit string:** Describe the changes you would make to X to become Y
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
- 0001
- 0000
Approximate Pattern Matching

\[ P = \text{abb} \quad d = 1 \]

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

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

A) \( \text{aba} \) 
B) \( \text{aabb} \)
C) \( \text{bbb} \) 
D) \( \text{ab} \)
Approximate Pattern Matching

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

$P$: 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
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!

\[ P \]

10111010111011010001010
Approximate Pattern Matching

Can we use our efficient exact pattern matching algorithms?
For Hamming distance (mismatches), we can!

\[ P \]

1011101011011011010001010

\[ P \]
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

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

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

**Pigeonhole principle:** A direct relationship between containers and objects from either perspective below.

$k+1$ pigeons, $k$ holes?

At least one hole has two pigeons!

$k$ pigeons, $k+1$ holes?

At least one hole is empty!
Approximate Pattern Matching

Pigeonhole principle lets us use exact matching algorithms:

$P$: word

$T$: There would have been a time for such a word
Approximate Pattern Matching

Pigeonhole principle lets us use exact matching algorithms:

\[ P: \text{word} \]
\[ T: \text{There would have been a time for such a word} \]
\[ u: \text{wo} \quad \text{wo} \quad \text{wo} \]
\[ v: \text{rd} \quad \text{rd} \]

1) Given \( k \) allowed mismatches, break the pattern up into \( k+1 \) partitions

What do we do with these partial matches?
Approximate Pattern Matching

\[ P: \text{word} \]

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

\[ u: \quad \text{wo} \quad \text{word} \quad \text{wo} \quad \text{word} \]

\[ v: \quad \text{rd} \quad \text{word} \]

1) Given \( k \) allowed mismatches, break the pattern up into \( k+1 \) partitions

2) Count mismatches in the remaining characters in the alignment

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

$P$: CATS

$T$: THE CART WAS CARRIED BY THE CATS
Approximate Pattern Matching

**Counting mismatches** requires verifying non-matching partitions

<table>
<thead>
<tr>
<th></th>
<th>(p_0)</th>
<th>(p_1)</th>
<th>(p_2)</th>
<th>(p_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>verify</td>
<td>verify</td>
<td>match</td>
<td>verify</td>
<td></td>
</tr>
</tbody>
</table>

\[ T \]

**CATS**

**THE CART WAS CARRIED BY THE CATS**
Approximate Pattern Matching

**Seed and Extend:** Using the pigeonhole principle to search a large text for exact matches and validating only these matches

Pros:

Cons:
Approximate Pattern Matching

$P$: AAAAAA
$T$: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

$P$: BBBAAB
$T$: AABAAAAAAAAAAAAAAAAABBAAAABAAAABAAAAAA
Approximate Pattern Matching

As a **heuristic**, seed and extend reduces the overall search space

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

\[ \text{word} \]

Only consider mismatches while verifying a seed hit
Approximate Pattern Matching

As a **heuristic**, seed and extend reduces the overall search space.

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

Consider the likelihood of seeing ‘wo’ or ‘rd’ **by chance**:

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

**Partition Seed**: Length ~40
CTCAAACTCCTGACCTTTGGTGATCCACCCGCCTAGGCCTTACCTGCATAATGAAG
AAGCACCCAACTTACACTTAGGAGATTTCAACTTAACTTGACCGCTCTGAGCTAAACCTA
GCCAAACCCACTCCACCTTACTACCAGACAACCTTAGCCAAACCATTTACCCAAATAA
AGTATAGGCGATAGAAATTGAAACCTGGCGCAATAGATATAGTACCGCAAGGGAAAGATG
AAAAATTATAACCAAGCATAATATAGCAAGGACTAACCCCTATACCTTCTGCATAATGAA
TTAACTAGAAATAACTTTGCAAGGAGAGCCAAAGCTAAGACCCCCGAAACCAGACGAGCT
ACCTAAGAACAGCTAAAAGAGCACACCCGTCTATGTAGCAAAATAGTGGGAAGATTTATA
GGTAGAGGCGACAAACCTACCGAGCCTGGTGATAGCTGGTTGTCCAAGATAGAATCTTAG
TTCAACTTTAAATTTGCCCACAGAACCCTCTAAATCCCCTTGTAAATTTAACTGTTAGTC
TTAAAGGAGACAGCTCTTTGGACACATAGAAAAACCTTGTAGAGAGAGTAAAAAATTTA

**T**: Length 3 billion
GATCACACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCATGCATTTGGTATTTT
CGTCTGCTGCTGCCGCTGATACGTTGGCAAGCTCTCTGAGCCCGACCCCTGCTCTCCTAC
GGATGATCTCTTTGATTCTGCTCCATCTATTGATGACATTTGCTATATATT
ACAGGCGAAATCTTAAGGTTTTAGTAAATGCCTGAACATTTGATGTTAAATAG
ACAATTGAAATGCTGACACGGACCTCCTCTCCTGACCATATACTTTTACATT
ACACCTGCTCCCTCCGCCGGCTTTCCCAACCAGCCCTTTAAAACATTT
AAAAAGACCCCTAAACCCCCCCCGCAACCCGGCGCTCTTGCATCGGCTGGGGGGTATGCAC
TCGCGACACGCGTATAGCATTGCGAGACGCTGGAGCCGGAGCACCCTATGTC
GCCGACGACAACGAACGCGGCCTTCTCCTCAACATGTCTGGGGGGTATGCACGCGATAGCG
CTTTTAAACGTCAACCCCCCACTACATTCTCCTCCTCCACCCCATCTACATTCTCATAT
CTCAGATACATACACCCCGTCCTTTACCCGAAACACCACTACGCTTTTACCCGAGCCCG
TGCTCAGAAGCTTACGAGCTTCCGCTACGCTTCCGCTACGCTTCCGCTACGCTTCCGCT
CCCGGCTAAAAACATAAAAGACCCCAACGCTGCTTTTAGCTGATATAGAATCTTAG
TTCAACTTTAAATTTGCCCACAGAACCCTCTAAATCCCCTTGTAAATTTAACTGTTAGTC
TTAAAGGAGACAGCTCTTTGGACACATAGAAAAACCTTGTAGAGAGAGTAAAAAATTTA

Likelihood of random seed string:

\[
\frac{1}{4^{\text{Length}}} = 8.27 \times 10^{-25}
\]

# of times seed will occur by chance in T:

\[
\text{Likelihood} \times (\text{~ length}) = 2.48 \times 10^{-15}
\]
Approximate Pattern Matching

“Seed and extend” approach to pattern matching

\[ p_0 \quad p_1 \quad p_2 \quad p_3 \quad \cdots \quad p_k \]

**Pros:**
- Reuse exact matching algs
- Works for Hamming and edit distance*
- As a heuristic, reduces search space

**Cons:**
- Slow for large \( k \)
- Small partitions matching many times by chance
- \( k+1 \) 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 $p_0, p_1, ..., p_k$ must appear with 0 mismatches.

$P = \text{A A A A A A A}$ \hspace{1cm} $d = 3$

How many partitions?

What is the characters in each partition(s)?

$T = \text{... B B B B B B A A B B B B B B ...}$
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.

$P = A B A C A A B A \quad d = 3$

How many partitions?

What is the characters in each partition(s)?

$T = \cdots B B B B A B A A B A B A B B B B \cdots$
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.

$P = \quad B \quad A \quad A \quad A \quad A \quad A$

$d = 1$

$T = \quad ... \quad B \quad B \quad B \quad B \quad B \quad B \quad A \quad A \quad A \quad A \quad A \quad B \quad A \quad A \quad ...$
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?
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: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
     aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
     aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa
     aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa aaaa

  k = 4
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} }
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
Bonus Slides
FM Index w/ mismatches

Start with shortest suffix, then match successively longer suffixes

Keep track of mismatches for each suffix

\[ P = \text{aba} \]

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

Easy to find all the rows beginning with \text{a}
FM Index w/ mismatches

We have rows beginning with $a$, now we want rows beginning with $ba$

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

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

$$P = \text{aba}$$

$$P = \text{aba}$$

Only works for Hamming Distance (mismatches)!