Boyer-Moore

Ben Langmead

Please sign guestbook (www.langmead-lab.org/teaching-materials) to tell me briefly how you are using the slides. For original Keynote files, email me (ben.langmead@gmail.com).
Can we improve on the naïve algorithm?

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

$u$ doesn’t occur in $P$, so skip next two alignments

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

word skip!
word skip!
word
Boyer-Moore

Learn from character comparisons to skip pointless alignments

1. When we hit a mismatch, move $P$ along until the mismatch becomes a match  “Bad character rule”
2. When we move $P$ along, make sure characters that matched in the last alignment also match in the next alignment “Good suffix rule”
3. Try alignments in one direction, but do character comparisons in opposite direction For longer skips

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

--- word ---

Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) $P$ moves past mismatched character. (c) If there was no mismatch, don't skip.
Boyer-Moore: Bad character rule

Step 1:

\[ T: \text{GCTTCTGCTA} \quad P: \text{CCCTTTGCGC} \]

Up to step 3, we skipped 8 alignments

5 characters in \( T \) were never looked at
Boyer-Moore: Good suffix rule

Let $t =$ substring matched by inner loop; skip until (a) there are no mismatches between $P$ and $t$ or (b) $P$ moves past $t$

Step 1:

$T$: CGTGCTTACCTTACTTACCTTACCTTACGCGAA

$P$: CTTACCTTAC

Step 2:

$T$: CGTGCCCTACTTTACCTTACTTACCTTACCTTACGCGAA

$P$: CTTACTTAC

Step 3:

$T$: CGTGCCCTACTCTTACTTACTTACTTACTTACGCGAA

$P$: CTTACTTAC
Boyer-Moore: Good suffix rule

Let $t =$ substring matched by inner loop; skip until (a) there are no mismatches between $P$ and $t$ or (b) $P$ moves past $t$

Step 1:

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

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

$t$ occurs *in its entirety* to the left within $P$

Step 2:

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

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

Prefix of $P$ matches a *suffix* of $t$

Step 3:

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

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

Case (a) has two subcases according to whether $t$ occurs *in its entirety* to the left within $P$ (as in step 1), or a prefix of $P$ matches a *suffix* of $t$ (as in step 2)
Boyer-Moore: Putting it together

How to combine bad character and good suffix rules?

\[ T: \text{GTTATAGCTGAT} \quad \text{bad char says skip 2, good suffix says skip 7} \]

\[ P: \text{GCGGCGTAGCGGCGAA} \quad \text{Take the maximum! (7)} \]
Boyer-Moore: Putting it together

Use bad character or good suffix rule, whichever skips more

Step 1:

\[ T: \text{GTTATAGC} \quad \text{P: GTAGCGGGC} \]

bc: 6, gs: 0  
bad character

Step 2:

\[ T: \text{GTTATAGCTGAT} \quad \text{P: GTAGCGGGC} \]

bc: 0, gs: 2  
good suffix

Step 3:

\[ T: \text{GTTATAGCTGATCGCGGCGTAGCGGCGAA} \quad \text{P: GTAGCGGGCGCG} \]

bc: 2, gs: 7  
good suffix

Step 4:

\[ T: \text{GTTATAGCTGATCGCGGCGTAGCGGCGAA} \quad \text{P: GTAGCGGGCGCG} \]
11 characters of $T$ we ignored

Step 1:
$T$: GTTATAGCTGATCGCGGCGTAGCGGCGAA
$P$: GTAGCGGGCG

Step 2:
$T$: GTTATAGCTGATCGCGGCGTAGCGGCGAA
$P$: GTAGCGGGCG

Step 3:
$T$: GTTATAGCTGATCGCGGCGTAGCGGCGAA
$P$: GTAGCGGGCG

Step 4:
$T$: GTTATAGCTGATCGCGGCGTAGCGGCGAA
$P$: GTAGCGGGCG

Skipped 15 alignments
Pre-calculate skips for all possible mismatch scenarios! For bad character rule and $P = TCGC$:
Boyer-Moore: Preprocessing

Pre-calculate skips for all possible mismatch scenarios!
For bad character rule and $P = TCGC$:

$$\Sigma$$

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>C</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

This can be constructed efficiently. See Gusfield 2.2.2.
Boyer-Moore: Preprocessing

As with bad character rule, good suffix rule skips can be precalculated efficiently. See Gusfield 2.2.4 and 2.2.5.

For both tables, the calculations only consider $P$. No knowledge of $T$ is required.

We’ll return to preprocessing soon!
Boyer-Moore: Good suffix rule

We learned the *weak* good suffix rule; there is also a *strong* good suffix rule.

Strong good suffix rule skips more than weak, at no additional penalty.

Strong rule is needed for proof of Boyer-Moore’s $O(n + m)$ worst-case time. Gusfield discusses proof(s) in first several sections of ch. 3.
Aside: Big-O notation

For review, see Jones & Pevzner 2.8

\[ O(n^2) \]

“big oh of n squared”

Asymptotic upper bound on worst-case growth
Boyer-Moore: Worst case

Boyer-Moore, with refinements in Gusfield, is $O(n + m)$ time

Given $n < m$, can simplify to $O(m)$

Is this better than naïve?

For naïve, worst-case # char comparisons is $n(m - n + 1)$

Boyer-Moore: $O(m)$, naïve: $O(nm)$

Reminder: $|P| = n$  $|T| = m$
Boyer-Moore: Best case

What’s the best case?

\( P: \quad \text{bbbb} \)

\( T: \quad \text{aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa} \)

\[
\begin{array}{cccccccc}
\text{bbbb} & \text{bbbb} & \text{bbbb} & \text{bbbb} & \text{bbbb} & \text{bbbb} & \text{bbbb} \\
\text{bbbb} & \text{bbbb} & \text{bbbb} & \text{bbbb} & \text{bbbb} & \text{bbbb} & \text{bbbb} \\
\end{array}
\]

Every alignment yields immediate mismatch and bad character rule skips \( n \) alignments

How many character comparisons? \( \text{floor}(m / n) \)
Naive vs Boyer-Moore

As $m \& n$ grow, # characters comparisons grows with...

| $|P| = n$ | $|T| = m$ | Naïve matching | Boyer-Moore |
|---------|---------|--------------|--------------|
| Worst case | $m \cdot n$ | $m$ |
| Best case | $m$ | $m / n$ |
Performance comparison

Simple Python implementations of naïve and Boyer-Moore:

<table>
<thead>
<tr>
<th></th>
<th>Naïve matching</th>
<th>Boyer-Moore</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># character comparisons</td>
<td>wall clock time</td>
</tr>
<tr>
<td><strong>P</strong>: “tomorrow”</td>
<td>5,906,125</td>
<td>2.90 s</td>
</tr>
<tr>
<td><strong>T</strong>: Shakespeare’s complete works</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>P</strong>: 50 nt string from Alu repeat*</td>
<td>307,013,905</td>
<td>137 s</td>
</tr>
<tr>
<td><strong>T</strong>: Human reference (hg19) chromosome 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

17 matches \( |T| = 5.59 \text{M} \)

336 matches \( |T| = 249 \text{M} \)

* GCGCGGTGGCTCACGCTGTAATCCCAGCACTTTGGGAGGGCCGAGGCGGG

* GCGCGGTGGCTCACGCTGTAATCCCAGCACTTTGGGAGGGCCGAGGCGGG
def boyer_moore(p, p_bm, t):
    """ Do Boyer-Moore matching """
    i = 0
    occurrences = []
    while i < len(t) - len(p) + 1:  # left to right
        shift = 1
        mismatched = False
        for j in range(len(p)-1, -1, -1):  # right to left
            if p[j] != t[i+j]:
                skip_bc = p_bm.bad_character_rule(j, t[i+j])
                skip_gs = p_bm.good_suffix_rule(j)
                shift = max(shift, skip_bc, skip_gs)
                mismatched = True
                break
        if not mismatched:
            occurrences.append(i)
            skip_gs = p_bm.match_skip()
            shift = max(shift, skip_gs)
            i += shift
    return occurrences
Preprocessing: Boyer-Moore

Make lookup tables for bad character & good suffix rules

Results
Preprocessing: Naïve algorithm

Naïve exact matching

Results
Preprocessing: Boyer-Moore

Preprocessing: trade one-time cost for reduced work overall via *reuse*

Boyer-Moore preprocesses $P$ into lookup tables that are *reused*

*reused* for each alignment of $P$ to $T_1$

If you later give me $T_2$, I *reuse* the tables to match $P$ to $T_2$

If you later give me $T_3$, I *reuse* the tables to match $P$ to $T_3$

...

Cost of preprocessing is *amortized* over alignments & texts