Hamming distance - considering symbols separately

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

$$T = \sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_5 \sigma_6 \sigma_7 \sigma_8 \sigma_9$$

$$P = \alpha_1 \alpha_2 \alpha_3 \alpha_4 \alpha_5 \alpha_6 \alpha_7 \alpha_8 \alpha_9$$

Replace all $$\alpha$$ symbols with 1 and everything else with 0

Hamming distance - considering symbols separately

Imagine that the alphabet contains only a small number of different symbols, which we consider individually...

$$T_n = \sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_5 \sigma_6 \sigma_7 \sigma_8 \sigma_9$$

$$P_n = \alpha_1 \alpha_2 \alpha_3 \alpha_4 \alpha_5 \alpha_6 \alpha_7 \alpha_8 \alpha_9$$

Replace all $$\alpha$$ symbols with 1 and everything else with 0

We denote these new strings $$T_n$$ and $$P_n$$ and consider...

$$(T_n \otimes P_n)_i = \sum_{j=0}^{m-1} P_n[j] T_n[i+j]$$

This is the number of matching $$\alpha$$s at the i-th alignment, which we can compute (for all i) in $$O(n \log m)$$ time via cross-correlations

Hamming distance - considering symbols separately

We saw how to find all matches with a single symbol in $$O(n \log m)$$ time

Let $$\Sigma$$ denote the set of alphabet symbols and $$|\Sigma|$$ be its size

**Algorithm Summary**

- Construct $$T_n$$ and $$P_n$$ for every symbol $$\sigma$$ in $$\Sigma$$ ($$O(n|\Sigma|)$$ time)
- Compute $$T_n \otimes P_n$$ (for every symbol $$\sigma$$ in $$\Sigma$$) ($$O(n|\Sigma| \log m)$$ time)

For every i, compute,

$$\text{Ham}(i) = m - \sum_{\sigma \in \Sigma} (T_n \otimes P_n)[i]. \quad (O(n|\Sigma|) \text{ time})$$

This takes $$O(n|\Sigma| \log m)$$ total time and $$O(n)$$ space

However, $$|\Sigma|$$ could be as big as $$m$$... what should we do instead?

The frequent/infrequent symbols trick

**Definition:** A symbol is frequent if it occurs at least $$\sqrt{m}$$ times in $$P$$.

$$P = \sigma_1 \sigma_2 \sigma_3 \sigma_4 \sigma_5 \sigma_6 \sigma_7 \sigma_8 \sigma_9$$

$$\alpha$$ is frequent, $$b$$ is frequent, $$c$$ and $$d$$ are not frequent

**Step 1:** Count all matches involving frequent symbols.

Consider each frequent symbol separately in $$O(n \log m)$$ time (per symbol), using cross-correlations

How many frequent symbols can there be?

Assume that there are at least $$\sqrt{m}$$ frequent symbols each occurs at least $$\sqrt{m}$$ times... ($$\sqrt{m} + 1)^2 > m$$ Contradiction!

so there are at most $$\sqrt{m}$$ frequent symbols

So Step 1 takes $$O(n \sqrt{m} \log m)$$ time.
The infrequent/frequent symbols trick

**Definition:** A symbol is **inrequent** if it occurs fewer than $\sqrt{m}$ times in $P$.

Every symbol is either frequent or infrequent $\alpha$ is frequent, $b$ is frequent $c$ and $d$ are infrequent.

**The infrequent/frequent symbols trick**

**Definition:** A symbol is *inrequent* if it occurs fewer than $\sqrt{m}$ times in $P$.

Every symbol is either frequent or infrequent:
- $\alpha$ is frequent,
- $b$ is frequent,
- $c$ and $d$ are infrequent.

<table>
<thead>
<tr>
<th>$T$</th>
<th>d</th>
<th>a</th>
<th>X</th>
<th>a</th>
<th>d</th>
<th>c</th>
<th>c</th>
<th>b</th>
<th>d</th>
<th>c</th>
<th>a</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>$A$</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Step 2:** Count all matches involving infrequent symbols.

Construct an array $A$ of length $(n - m + 1)$ which is initially all zeros.

- Make a single pass through $T$.
- For each character $T[k]$, where $0 \leq k < n$.
  - If $T[k]$ is infrequent:
    - Increase $A[k - j]$ by one for all $j$ such that $T[k] = P[j]$.
    - Store a list for each infrequent symbol.
    - Each list has length less than $\sqrt{m}$.

Overall, we obtain a time complexity of $O(n\sqrt{m} \log m)$. 

Pattern matching with mismatches: putting it all together

**Algorithm summary**

**Step 0:** Classify each symbol as frequent or infrequent ($O(m \log m)$ time)

**Step 1:** Count all matches involving frequent symbols. ($O(n\sqrt{m} \log m)$ time)

**Step 2:** Count all matches involving infrequent symbols. ($O(n\sqrt{m} \log m)$ time)

at any alignment the number of mismatches is just $m$ minus the total number of matches.

Overall, we obtain a time complexity of $O(n\sqrt{m} \log m)$.

Pattern matching with few mismatches ($k$-mismatch)

**Input** A text string $T$ (length $n$), a pattern string $P$ (length $m$) and a positive integer $k$

<table>
<thead>
<tr>
<th>$T$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>d</th>
<th>a</th>
<th>c</th>
<th>c</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>$A$</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\text{Ham}_k(8) = X$

**Goal:** For all $i$, output:

<table>
<thead>
<tr>
<th>$\text{Ham}_k(i)$</th>
<th>$\text{Ham}(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i &lt; k$</td>
<td>$\text{Ham}(i)$</td>
</tr>
<tr>
<td>$i &gt; k$</td>
<td>$X$</td>
</tr>
</tbody>
</table>

Output the number of mismatches... unless it's more than $k$ (we interpret the output $X$ to mean ‘too many mismatches’)

- We could use the $O(n\sqrt{m} \log m)$ time algorithm for Hamming distance... but when $k$ is small we can do much better
LCP - the Longest Common Prefix

For any pair of locations $i$ in $T$ and $j$ in $P$, $\text{LCP}(i, j)$ is the largest $d$ such that $T[i \ldots i + d - 1] = P[j \ldots j + d - 1]$

it's the furthest you can go before hitting a mismatch

For any pair of locations $i$ in $T$ and $j$ in $P'$, $\text{LCP}(i, j)$ is the largest $d$ such that $T[i \ldots i + d - 1] = P'[j \ldots j + d - 1]$

it's the furthest you can go before hitting a mismatch

---

Fact: We'll see that $\text{LCP}$ queries can be used to calculate $\text{Ham}_k(i)$ in $O(k)$ time

---

Algorithm summary

We can use $\text{LCP}$ queries to solve $k$-mismatch, as well as $\text{Ham}_k$...

---

$k$-mismatch using frequent/infrequent symbols

Definition: A symbol is frequent if it occurs at least $\sqrt{k}$ times in $P$, and infrequent otherwise

$$k = 4 \quad (\sqrt{k} = 2)$$

$a$ is frequent, $b$ is frequent, $d$ is frequent
$c$ is infrequent

How many frequent symbols can there be? Lots! there could be $\frac{2k}{\sqrt{k}} = \sqrt{k}$ frequent symbols

---

Case 1: There are fewer than $2\sqrt{k}$ frequent symbols in $P$. - $O(n\sqrt{k} \log m)$ total time

Algorithm summary

Step 0: Classify each symbol as frequent or infrequent - $O(m \log m)$ time

Step 1: Count all matches involving frequent symbols (using convolutions) - $O(n\sqrt{k} \log m)$ time

Step 2: Count all matches involving infrequent symbols (as before) - $O(n\sqrt{k})$ time

---

Case 2: There are at least $2\sqrt{k}$ frequent symbols

Pick any $2\sqrt{k}$ frequent symbols and for each symbol pick $\sqrt{k}$ occurrences in $P$.

This gives us $2k$ interesting pattern locations, denoted $J$

$$J = \{0, 2, 3, 4, 5, 7, 9, 10\}$$

$k = 4$

Let $d_k(i)$ be the number of $j \in J$ such that $P[j] = T[i + j]$

i.e. the number of (single character) matches involving interesting pattern locations

Fact: If $d_k(i) < k$ then there are more than $k$ mismatches (i.e. $\text{Ham}_k(i) = X$) because there are $2k$ interesting positions... and fewer than $k$ of them match

Fact: There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$

this follows from a counting argument
Case 2: There are at least $2\sqrt{k}$ frequent symbols

Pick any $2\sqrt{k}$ frequent symbols and for each symbol pick $\sqrt{k}$ occurrences in $P$.

This gives us $2k$ interesting pattern locations, denoted $J$.

$$J = \{0, 2, 3, 4, 5, 7, 9, 10\}$$

$k = 4$

$P$

\[\begin{array}{cccccccccc}
\text{a} & \text{b} & \text{b} & \text{a} & \text{c} & \text{d} & \text{b} & \text{d} & \text{f} & \text{b} & \text{d} \\
\hline
\text{a} & \text{c} & \text{c} & \text{a} & \text{b} & \text{a} & \text{c} & \text{d} & \text{f} & \text{f} & \text{b} & \text{c}
\end{array}\]

$i = 4$

$d_k(i) = 3$

Let $d_k(i)$ be the number of $j \in J$ such that $P[j] = T[i + j]$

i.e. the number of (single character) matches involving interesting pattern locations

**Fact** There are at most $n/\sqrt{k}$ values of $i$ with $d_k(i) \geq k$

For any location $i'$, $T[i'] = P[j]$ for either 0 or $\sqrt{k}$ distinct $j \in J$

This implies that $\sum_i d_k(i) \leq \sum_{i'} \sum_{j \in J} \text{Eq}(T[i'], P[j]) \leq n/\sqrt{k}$

**Algorithm summary**

Preprocess $P$, $T$ for LCP queries - $O(n)$ time

Count the number of frequent symbols in $P$ - $O(m \log m)$ time

**Case 1:** $P$ has at most $2\sqrt{k}$ frequent symbols

Count matches with frequent symbols using convolution - $O(n/\sqrt{k} \log m)$ time

Count matches with infrequent symbols directly - $O(n/\sqrt{k})$ time

**Case 2:** $P$ has more than $2\sqrt{k}$ frequent symbols

Filter the text, leaving $n/\sqrt{k}$ alignments - $O(n/\sqrt{k})$ time

Count mismatches at these alignments using LCP queries - $O(n/\sqrt{k})$ time

Overall, we obtain a time complexity of $O(n/\sqrt{k} \log m)$.

- This can be improved to $O(n/\sqrt{k} \log k)$