The Fine-Grained Complexity of Episode Matching

June 24, 2022

Philip Bille, Inge Li Gørtz, Shay Mozes, Teresa Anna Steiner, Oren Weimann
Episode Matching

\[ P = \text{ANANAS} \]

\[ S = \text{BATMAN AND ANNA SING NANANANA AND EAT BANANAS} \]
Episode Matching

\[ P = \text{ANANAS} \]

\[ S = \text{BATMAN AND ANNA SING NANANANA AND EAT BANANAS} \]

- Find minimal substrings of \( S \) containing \( P \) as a subsequence
Episode Matching

$P = \text{ANANAS}$

$S = \text{BATMAN AND ANNA SING NANANA AND EAT BANANAS}$

- Find minimal substrings of $S$ containing $P$ as a subsequence
- The minimal substrings of $S$ which contain $P$ as a subsequence are shown in blue: $S[6, 16]$ and $S[39, 44]$
$P = \text{ANANAS}$

$S = \text{BATMAN AND ANNA SING NANANA ANNA AND EAT BANANAS}$

- Find minimal substrings of $S$ containing $P$ as a subsequence
- The minimal substrings of $S$ which contain $P$ as a subsequence are shown in blue: $S[6, 16]$ and $S[39, 44]$.
- We consider a version of the problem where the goal is to find the length of the shortest substring of $S$ containing $P$ as a subsequence.
Complexities - Algorithms

- $|P| = m$, $|S| = n$
Complexities - Algorithms

- $|P| = m$, $|S| = n$
- (Old) upper bound: $O(nm/\log n)$ (Das et al. [DFG+97])
• $|P| = m$, $|S| = n$

• (Old) upper bound: $O(nm/\log n)$ (Das et al. [DFG+97])

• **This work:** no $O(nm^{1-\epsilon})$ or $O(n^{1-\epsilon}m)$ algorithm assuming OVH
Our OV reduction + Equi et al. [EMT21]: polynomial time preprocessing does not help
Complexities - Data structures

- Our OV reduction + Equi et al. [EMT21]: polynomial time preprocessing does not help
- Time/Space tradeoffs:
Complexities - Data structures

- Our OV reduction + Equi et al. [EMT21]: polynomial time preprocessing does not help
- Time/Space tradeoffs:

\[
O(n) \quad O(k \cdot \tau \cdot \log \log n)
\]

\[
O(n) \quad O(k \cdot \tau \cdot \log \log n)
\]

\[
\Omega(nk - k\delta - o(1)) \quad O(n\delta)
\]

\[
m = k \text{ fixed}
\]

- \(\text{dist}_{occ}(P_1 \ldots P_i)\) is the number of distinct minimal substrings containing \(P_1 \ldots P_i\) as a subsequence

- Conditional lower bound based on hardness of \(k\)-Set Disjointness
• Our OV reduction + Equi et al. [EMT21]: polynomial time preprocessing does not help
• Time/Space tradeoffs:

<table>
<thead>
<tr>
<th></th>
<th>space</th>
<th>time</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[AA02]</td>
<td>$O(n)$</td>
<td>$O(\sum_{i=1}^{m} \text{dist}_\text{occ}(P_i) \cdot i)$</td>
<td></td>
</tr>
<tr>
<td>This work</td>
<td>$O(n + \left(\frac{n}{\tau}\right)^k)$</td>
<td>$O(k \cdot \tau \cdot \log \log n)$</td>
<td>$m = k$ fixed</td>
</tr>
<tr>
<td>This work</td>
<td>$\Omega(n^{k-k\delta-o(1)})$</td>
<td>$O(n^\delta)$</td>
<td>$m = k$ fixed</td>
</tr>
</tbody>
</table>
Complexities - Data structures

- Our OV reduction + Equi et al. [EMT21]: polynomial time preprocessing does not help
- Time/Space tradeoffs:

<table>
<thead>
<tr>
<th></th>
<th>space</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[AA02]</td>
<td>(O(n))</td>
<td>(O(\sum_{i=1}^{m} dist_occ(P_i) \cdot i))</td>
</tr>
<tr>
<td>This work</td>
<td>(O(n + \left(\frac{n}{\tau}\right)^k))</td>
<td>(O(k \cdot \tau \cdot \log \log n))</td>
</tr>
<tr>
<td>This work</td>
<td>(\Omega(n^{k-k\delta-o(1)}))</td>
<td>(O(n^\delta))</td>
</tr>
</tbody>
</table>

- \(dist\_occ(P_i)\) is the number of distinct minimal substrings containing \(P[1] \ldots P[i]\) as a subsequence
Complexities - Data structures

- Our OV reduction + Equi et al. [EMT21]: polynomial time preprocessing does not help
- Time/Space tradeoffs:

<table>
<thead>
<tr>
<th></th>
<th>space</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>[AA02]</td>
<td>$O(n)$</td>
<td>$O(\sum_{i=1}^{m} \text{dist_occ}(P_i) \cdot i)$</td>
</tr>
<tr>
<td>This work</td>
<td>$O(n + (\frac{n}{\tau})^k)$</td>
<td>$O(k \cdot \tau \cdot \log \log n)$</td>
</tr>
<tr>
<td>This work</td>
<td>$\Omega(n^{k - k\delta - o(1)})$</td>
<td>$O(n^\delta)$</td>
</tr>
</tbody>
</table>

- $\text{dist\_occ}(P_i)$ is the number of distinct minimal substrings containing $P[1] \ldots P[i]$ as a subsequence
- Conditional lower bound based on hardness of $k-$Set Disjointness
• **This work:** Faster preprocessing for decision version using min-plus matrix multiplication
Orthogonal Vectors

- Two sets $A, B$ of $d$-dimensional, binary vectors, each set has size $n$
- Problem: Decide if there is a vector in $A$ that is orthogonal to a vector in $B$
- OVH: There is no algorithm running in time $O(n^{2-\epsilon}\text{poly}(d))$
• build $P$ from $B$: for $b \in B$, separate each coordinate by new letter $x$
  eg: $101 \rightarrow 1x0x1$
OV → Episode Matching

- build $P$ from $B$: for $b \in B$, separate each coordinate by new letter $x$
  eg: $101 \rightarrow 1x0x1$
- concatenate and separate by new letter $y$
  eg: $B = \{101, 111, 110\}$, $P = 1x0x1y1x1x1y1x1x0$
• build $P$ from $B$: for $b \in B$, separate each coordinate by new letter $x$
  eg: $101 \rightarrow 1x0x1$
• concatenate and separate by new letter $y$
  eg: $B = \{101, 111, 110\}$,
  $P = 1x0x1y1x1x1y1x1x0$
• Length of $P = O(nd)$
OV → Episode Matching

- build $S$ from $A$: for $a \in A$,
  
  $0 \rightarrow 01$

  $1 \rightarrow 00$

  separate each coordinate by letter $x$

  eg: $100 \rightarrow 00x01x01$
Episode Matching

- build $S$ from $A$: for $a \in A$,

$$
0 \rightarrow 01 \\
1 \rightarrow 00
$$

separate each coordinate by letter $\times$

eg: $100 \rightarrow 00\times01\times01$
OV → Episode Matching

- build $S$ from $A$: for $a \in A$,

  
  $0 \rightarrow 01$  
  $1 \rightarrow 00$

  separate each coordinate by letter $x$

  eg: $100 \rightarrow 00x01x01$

- let $b_1 = 010$, $b_2 = 110$, $a = 100$
• build $S$ from $A$: for $a \in A$,

\[
0 \rightarrow 01 \\
1 \rightarrow 00
\]

separate each coordinate by letter $\times$

eg: $100 \rightarrow 00\times01\times01$

• let $b_1 = 010$, $b_2 = 110$, $a = 100$

• $b_1$ and $a$ are orthogonal
OV→ Episode Matching

- build $S$ from $A$: for $a \in A$,

\[
0 \rightarrow 01 \\
1 \rightarrow 00
\]

separate each coordinate by letter $\times$

eg: $100 \rightarrow 00\times01\times01$

- let $b_1 = 010$, $b_2 = 110$, $a = 100$

- $b_1$ and $a$ are orthogonal

- $0\times1\times0$ is a subsequence of $00\times01\times01$
OV→ Episode Matching

- build $S$ from $A$: for $a \in A$,
  
  $0 \rightarrow 01$
  
  $1 \rightarrow 00$

  separate each coordinate by letter $x$

  eg: $100 \rightarrow 00x01x01$

- let $b_1 = 010$, $b_2 = 110$, $a = 100$

- $b_1$ and $a$ are orthogonal

- $0x1x0$ is a subsequence of $00x01x01$

- $b_2$ and $a$ are not orthogonal
OV→ Episode Matching

- build $S$ from $A$: for $a \in A$,
  
  $0 \rightarrow 01$
  
  $1 \rightarrow 00$

  separate each coordinate by letter $x$
  
  eg: $100 \rightarrow 00x01x01$

- let $b_1 = 010$, $b_2 = 110$, $a = 100$

- $b_1$ and $a$ are orthogonal

- $0x1x0$ is a subsequence of $00x01x01$

- $b_2$ and $a$ are not orthogonal

- $1x1x0$ is not a subsequence of $00x01x01$
### OV → Episode Matching

<table>
<thead>
<tr>
<th>a</th>
<th>s(a)</th>
<th>b</th>
<th>p(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>01×00×01</td>
<td>010</td>
<td>0×1×0</td>
</tr>
</tbody>
</table>

- to build $S$ as follows:
Episode Matching

<table>
<thead>
<tr>
<th>a</th>
<th>s(a)</th>
<th>b</th>
<th>p(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>01×00×01</td>
<td>010</td>
<td>0×1×0</td>
</tr>
</tbody>
</table>

- to build $S$ as follows:
- let $z$ be the $d$-dimensional 0 vector
Episode Matching

<table>
<thead>
<tr>
<th></th>
<th>s(a)</th>
<th>b</th>
<th>p(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>010</td>
<td>010</td>
<td>0x1x0</td>
</tr>
<tr>
<td>01x00x01</td>
<td></td>
<td>010</td>
<td>0x1x0</td>
</tr>
</tbody>
</table>

- to build $S$ as follows:
- let $z$ be the $d$-dimensional 0 vector
- $s(z) = 01x01x \ldots x01$
Episode Matching

<table>
<thead>
<tr>
<th>a</th>
<th>s(a)</th>
<th>b</th>
<th>p(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>01*00*01</td>
<td>010</td>
<td>0*1*0</td>
</tr>
</tbody>
</table>

- to build $S$ as follows:
- let $z$ be the $d$-dimensional 0 vector
- $s(z) = 01\*01\*01\ldots 01$
- $S = s(a_1)ys(z)ys(a_2)ys(z)y\ldots s(a_n)ys(z)ys(a_1)ys(z)y\ldots s(n)$
to build $S$ as follows:

- let $z$ be the $d$-dimensional 0 vector
- $s(z) = 01x01x \ldots x01$
- $S = s(a_1)ys(z)ys(a_2)ys(z)y \ldots s(a_n)ys(z)ys(a_1)ys(z)y \ldots s(n)$

- Length of $S = O(nd)$
to build $S$ as follows:

- let $z$ be the $d$-dimensional 0 vector
- $s(z) = 01x01x \ldots x01$
- $S = s(a_1)ys(z)ys(a_2)ys(z)ys(a_3)ys(z) \ldots s(a_n)ys(z)ys(a_1)ys(z)ys(a_2)ys(z) \ldots s(n)$
- Length of $S = O(nd)$
- $|P| |S|^{1-\epsilon} = O(n^{2-\epsilon} d^{2-\epsilon})$
- $|P|^{1-\epsilon} |S| = O(n^{2-\epsilon} d^{2-\epsilon})$
**Episode Matching**

<table>
<thead>
<tr>
<th>a</th>
<th>s(a)</th>
<th>b</th>
<th>p(b)</th>
<th>z</th>
<th>s(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>01x00x01</td>
<td>010</td>
<td>0x1x0</td>
<td>000</td>
<td>01x01x01</td>
</tr>
</tbody>
</table>

No orthogonal vectors:

\[
\begin{align*}
  y & s(z) & y & s(a_{i-1}) & y & s(z) & y & s(a_i) & y & s(z) & y & s(a_{i+1}) \\
  y & p(b_{j-1}) & y & p(b_j) & y & p(b_{j+1})
\end{align*}
\]
OV→ Episode Matching

$a_i, b_j$ orthogonal:

\[
\begin{align*}
&y \ s(z) \ y \ s(a_{i-1}) \ y \ s(z) \ y \ s(a_i) \ y \ s(z) \ y \ s(a_{i+1}) \\
&y \ p(b_{j-2}) \ y \ p(b_{j-1}) \ y \ p(b_j) \ y \ p(b_{j+1})
\end{align*}
\]
OV → Episode Matching

\[ S = s(\alpha_1) y s(\beta_1) y s(\alpha_2) y s(\beta_2) y \ldots s(\alpha_n) y s(\beta_n) y s(\alpha_1) y s(\beta_1) y \ldots s(\alpha_n) \]

- \( a_i \perp b_j \)
- \( j < i \): “overflow” to the right
- \( j > i \): “overflow” to the left
Equation: \[ S = s(a_1)ys(z)ys(a_2)ys(z)y \ldots s(a_n)ys(z)ys(a_1)ys(z)y \ldots s(a_n) \]

- \( a_i \perp b_j \)
- \( j < i \): “overflow” to the right
- \( j > i \): “overflow” to the left
$S = s(a_1)ys(z)ys(a_2)ys(z)y \ldots s(a_n)ys(z)ys(a_1)ys(z)y \ldots s(a_n)$

- $a_i \perp b_j$
- $j < i$: “overflow” to the right
- $j > i$: “overflow” to the left
• replace $x$ and $y$ by binary gadgets
• $|P| = k$ fixed at preprocessing

• Upper bound: Space: $O(n + \left(\frac{n}{\tau}\right)^k)$, Time: $O(k \cdot \tau \cdot \log \log n)$

• Conditional lower bound: Space: $\Omega(n^{k-k\delta-o(1)})$, Time: $O(n^\delta)$
Definition (*k*-Set Disjointness Problem)
Preprocess *m* sets $S_1, S_2, \ldots, S_m$ of total size $\sum_{i=1}^m |S_i| = N$ drawn from a universe $U$ such that given $(i_1, i_2, \ldots, i_k)$ we can quickly decide whether $\bigcap_{j=1}^k S_{i_j} = \emptyset$. 

• Up to log $N$ factors equivalent to the problem where every element appears in the same number of sets [BGPS21]

Conjecture (Strong *k*-Set Disjointness Conjecture)
Any data structure for the *k*-Set Disjointness Problem that answers queries in time $T$ must use $\widetilde{\Omega}(N^{k/T})$ space.
**Definition (k-Set Disjointness Problem)**
Preprocess $m$ sets $S_1, S_2, \ldots, S_m$ of total size $\sum_{i=1}^{m} |S_i| = N$ drawn from a universe $U$ such that given $(i_1, i_2, \ldots, i_k)$ we can quickly decide whether $\bigcap_{j=1}^{k} S_{i_j} = \emptyset$.

- Up to $\log N$ factors equivalent to the problem where every element appears in the same number of sets [BGPS21]
Definition (**k-Set Disjointness Problem**)
Preprocess $m$ sets $S_1, S_2, \ldots, S_m$ of total size $\sum_{i=1}^{m} |S_i| = N$ drawn from a universe $U$ such that given $(i_1, i_2, \ldots, i_k)$ we can quickly decide whether $\bigcap_{j=1}^{k} S_{i_j} = \emptyset$.

- Up to log $N$ factors equivalent to the problem where every element appears in the same number of sets [BGPS21]

Conjecture (**Strong k-Set Disjointness Conjecture**)

Any data structure for the $k$-Set Disjointness Problem that answers queries in time $T$ must use $\tilde{\Omega}(N^k / T^k)$ space.
Space/time trade-off, Lower bound

\[ S_1 = \{1, 3, 4\} \quad \alpha_1 \]
\[ S_2 = \{2\} \quad \alpha_2 \]
\[ S_3 = \{1, 2, 3, 4\} \quad \alpha_3 \]
\[ S_4 = \{2, 4\} \quad \alpha_4 \]
\[ S_5 = \{1, 3\} \quad \alpha_5 \]
Space/time trade-off, Lower bound

\[ S_1 = \{1, 3, 4\} \quad \alpha_1 \]
\[ S_2 = \{2\} \quad \alpha_2 \]
\[ S_3 = \{1, 2, 3, 4\} \quad \alpha_3 \]
\[ S_4 = \{2, 4\} \quad \alpha_4 \]
\[ S_5 = \{1, 3\} \quad \alpha_5 \]

\[ S = \underbrace{\alpha_1 \alpha_3 \alpha_5}_{1} \quad \underbrace{\alpha_2 \alpha_3 \alpha_4}_{2} \quad \underbrace{\alpha_1 \alpha_3 \alpha_5}_{3} \quad \underbrace{\alpha_1 \alpha_3 \alpha_4}_{4} \]
Space/time trade-off, Lower bound

\[ S_1 = \{1, 3, 4\} \quad \alpha_1 \]
\[ S_2 = \{2\} \quad \alpha_2 \]
\[ S_3 = \{1, 2, 3, 4\} \quad \alpha_3 \]
\[ S_4 = \{2, 4\} \quad \alpha_4 \]
\[ S_5 = \{1, 3\} \quad \alpha_5 \]

\[ S = \underbrace{\alpha_1 \alpha_3 \alpha_5} \quad \underbrace{\alpha_2 \alpha_3 \alpha_4} \quad \underbrace{\alpha_1 \alpha_3 \alpha_5} \quad \underbrace{\alpha_1 \alpha_3 \alpha_4} \]

\[ S_1 \cap S_4 = \emptyset? \]
Space/time trade-off, Lower bound

\[ S_1 = \{1, 3, 4\} \quad \alpha_1 \]
\[ S_2 = \{2\} \quad \alpha_2 \]
\[ S_3 = \{1, 2, 3, 4\} \quad \alpha_3 \]
\[ S_4 = \{2, 4\} \quad \alpha_4 \]
\[ S_5 = \{1, 3\} \quad \alpha_5 \]

\[ S = \underbrace{\alpha_1 \alpha_3 \alpha_5}_{1} \quad $$$ \quad \underbrace{\alpha_2 \alpha_3 \alpha_4}_{2} \quad $$$ \quad \underbrace{\alpha_1 \alpha_3 \alpha_5}_{3} \quad $$$ \quad \underbrace{\alpha_1 \alpha_3 \alpha_4}_{4} \]

\[ S_1 \cap S_4 = \emptyset? \]
\[ P_1 = \alpha_1 \alpha_4 \]
Space/time trade-off, Lower bound

\[ S_1 = \{1, 3, 4\} \quad \alpha_1 \]
\[ S_2 = \{2\} \quad \alpha_2 \]
\[ S_3 = \{1, 2, 3, 4\} \quad \alpha_3 \]
\[ S_4 = \{2, 4\} \quad \alpha_4 \]
\[ S_5 = \{1, 3\} \quad \alpha_5 \]

\[ S = \alpha_1 \alpha_3 \alpha_5 \quad $$$ \quad \alpha_2 \alpha_3 \alpha_4 \quad $$$ \quad \alpha_1 \alpha_3 \alpha_5 \quad $$$ \quad \alpha_1 \alpha_3 \alpha_4 \]

\[ S_1 \cap S_4 = \emptyset? \]
\[ P_1 = \alpha_1 \alpha_4 \]
\[ S_2 \cap S_5 = \emptyset? \]
Space/time trade-off, Lower bound

\[ S_1 = \{1, 3, 4\} \quad \alpha_1 \]
\[ S_2 = \{2\} \quad \alpha_2 \]
\[ S_3 = \{1, 2, 3, 4\} \quad \alpha_3 \]
\[ S_4 = \{2, 4\} \quad \alpha_4 \]
\[ S_5 = \{1, 3\} \quad \alpha_5 \]

\[ S = \underbrace{\alpha_1 \alpha_3 \alpha_5}_{1} \quad \underbrace{\alpha_2 \alpha_3 \alpha_4}_{2} \quad \underbrace{\alpha_1 \alpha_3 \alpha_5}_{3} \quad \underbrace{\alpha_1 \alpha_3 \alpha_4}_{4} \]

\[ S_1 \cap S_4 = \emptyset? \]
\[ P_1 = \alpha_1 \alpha_4 \]
\[ S_2 \cap S_5 = \emptyset? \]
\[ P_2 = \alpha_2 \alpha_5 \]
Space/time trade-off, upper bound

- Space=$O(n + (\frac{n}{\tau})^k)$, Time= $O(k \cdot \tau \cdot \log \log n)$
Space/time trade-off, upper bound

- Space=$O(n + \left(\frac{n}{\tau}\right)^k)$, Time=$O(k \cdot \tau \cdot \log \log n)$
- Call letters appearing more than $\tau$ times frequent
Space/time trade-off, upper bound

- Space=$O(n + \left(\frac{n}{\tau}\right)^k)$, Time= $O(k \cdot \tau \cdot \log \log n)$
- Call letters appearing more than $\tau$ times frequent
- For all $k$–tuples of frequent letters precompute answers
Space/time trade-off, upper bound

- Space = $O(n + (\frac{n}{\tau})^k)$, Time = $O(k \cdot \tau \cdot \log \log n)$
- Call letters appearing more than $\tau$ times frequent
- For all $k$–tuples of frequent letters precompute answers
- Have a predecessor data structure for each letter (total size = $O(n)$)
Space/time trade-off, upper bound

- Space = $O(n + \left(\frac{n}{\tau}\right)^k)$, Time = $O(k \cdot \tau \cdot \log \log n)$
- Call letters appearing more than $\tau$ times frequent
- For all $k$–tuples of frequent letters precompute answers
- Have a predecessor data structure for each letter (total size = $O(n)$)
- If $P$ contains non-frequent letter, “brute-force” using predecessor / successor
Space/time trade-off, upper bound

- Space = $O(n + \left(\frac{n}{\tau}\right)^k)$, Time = $O(k \cdot \tau \cdot \log \log n)$
- Call letters appearing more than $\tau$ times frequent
- For all $k$-tuples of frequent letters precompute answers
- Have a predecessor data structure for each letter (total size = $O(n)$)
- If $P$ contains non-frequent letter, “brute-force” using predecessor / successor
Space/time trade-off, upper bound

- Space = $O(n + \left(\frac{n}{\tau}\right)^k)$, Time = $O(k \cdot \tau \cdot \log \log n)$
- Call letters appearing more than $\tau$ times frequent
- For all $k$-tuples of frequent letters precompute answers
- Have a predecessor data structure for each letter (total size = $O(n)$)
- If $P$ contains non-frequent letter, “brute-force” using predecessor / successor

$P = \text{ANANAS}$

$S = \text{BATMAN AND ANNA SING NANANA AND EAT BANANAS}$
Space/time trade-off, upper bound

- Space\(=O(n + (\frac{n}{\tau})^k)\), Time\(= O(k \cdot \tau \cdot \log \log n)\)
- Call letters appearing more than \(\tau\) times frequent
- For all \(k\)–tuples of frequent letters precompute answers
- Have a predecessor data structure for each letter (total size = \(O(n)\))
- If \(P\) contains non-frequent letter, “brute-force” using predecessor / successor

\[ P = \text{ANANAS} \]
\[ S = \text{BATMAN AND ANNA SING NANANANA AND EAT BANANAS} \]

0 5 10 15 20 25 30 35 40
Space/time trade-off, upper bound

- Space = $O(n + \left(\frac{n}{\tau}\right)^k)$, Time = $O(k \cdot \tau \cdot \log \log n)$
- Call letters appearing more than $\tau$ times frequent
- For all $k$-tuples of frequent letters precompute answers
- Have a predecessor data structure for each letter (total size = $O(n)$)
- If $P$ contains non-frequent letter, “brute-force” using predecessor / successor

$P = \text{ANANAS}$

$S = \text{BATMAN AND ANNA SING NANANANA AND EAT BANANAS}$
Space/time trade-off, upper bound

- Space=\(O(n + \left(\frac{n}{\tau}\right)^k\)), Time=\(O(k \cdot \tau \cdot \log \log n)\)
- Call letters appearing more than \(\tau\) times frequent
- For all \(k\)-tuples of frequent letters precompute answers
- Have a predecessor data structure for each letter (total size = \(O(n)\))
- If \(P\) contains non-frequent letter, “brute-force” using predecessor / successor

\[P = \text{ANANAS}\]

\[S = \text{BATMAN AND ANNA SING NANANANA AND EAT BANANAS}\]

\[
\begin{array}{cccccccccccc}
0 & 5 & 10 & 15 & 20 & 25 & 30 & 35 & 40 \\
\end{array}
\]
Space/time trade-off, upper bound

• Space=$O(n + (\frac{n}{\tau})^k)$, Time= $O(k \cdot \tau \cdot \log \log n)$
• Call letters appearing more than $\tau$ times frequent
• For all $k$–tuples of frequent letters precompute answers
• Have a predecessor data structure for each letter (total size = $O(n)$)
• If $P$ contains non-frequent letter, “brute-force” using predecessor / successor

$P = \text{ANANAS}$

$S = \text{BATMAN AND ANNA SING NANANANA AND EAT BANANAS}$
Space/time trade-off, upper bound

- Space=$O(n + \left(\frac{n}{\tau}\right)^k)$, Time=$O(k \cdot \tau \cdot \log \log n)$
- Call letters appearing more than $\tau$ times frequent
- For all $k$–tuples of frequent letters precompute answers
- Have a predecessor data structure for each letter (total size = $O(n)$)
- If $P$ contains non-frequent letter, “brute-force” using predecessor / successor

$P = \text{ANANAS}$

$S = \text{BATMAN AND ANNA SING NANANA ANA AND EAT BANANAS}$

<table>
<thead>
<tr>
<th></th>
<th>BATMAN</th>
<th>AND</th>
<th>ANNA</th>
<th>SING</th>
<th>NANANA</th>
<th>ANA</th>
<th>AND</th>
<th>EAT</th>
<th>BANANAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Thank you!

Contact: teresa.anna.steiner@univie.ac.at
Alberto Apostolico and Mikhail J. Atallah. 
**Compact recognizers of episode sequences.**

Philip Bille, Inge Li Gørtz, Max Rishøj Pedersen, and Teresa Anna Steiner. 
**Gapped indexing for consecutive occurrences.**

Gautam Das, Rudolf Fleischer, Leszek Gasieniec, Dimitrios Gunopulos, and Juha Kärkkäinen. 
**Episode matching.**

Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. 
**Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails.**