The Fine-Grained Complexity of Episode Matching

June 24, 2022

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

Episode Matching

$P = \texttt{ANANAS}$

$S = \underset{0}{\texttt{B}}\texttt{ATMA}\underset{5}{\texttt{N}}\texttt{ AND}\underset{10}{\texttt{ }}\texttt{ANNA}\underset{15}{\texttt{ }}\texttt{SING}\underset{20}{\texttt{ }}\texttt{NANA}\underset{25}{\texttt{N}}\texttt{ANA }\underset{30}{\texttt{A}}\texttt{ND E}\underset{35}{\texttt{A}}\texttt{T BA}\underset{40}{\texttt{N}}\texttt{ANAS}$

Episode Matching

$P = \texttt{ANANAS}$

$S = \underset{0}{\texttt{B}}\texttt{ATMA}\underset{5}{\texttt{N}}\texttt{ AND }\underset{10}{\texttt{A}}\texttt{NNA }\underset{15}{\texttt{S}}\texttt{ING }\underset{20}{\texttt{N}}\texttt{ANAN}\underset{25}{\texttt{A}}\texttt{NA A}\underset{30}{\texttt{N}}\texttt{D EA}\underset{35}{\texttt{T}}\texttt{ BAN}\underset{40}{\texttt{A}}\texttt{NAS}$

- Find minimal substrings of S containing P as a subsequence

Episode Matching

$P =$ `ANANAS`

$S =$ `BATMAN AND ANNA SING NANANANA AND EAT BANANAS`

```
BATMAN AND ANNA SING NANANANA AND EAT BANANAS
0    5    10   15   20   25   30   35   40
```

- 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 $S[39, 44]$

Episode Matching

$P = \texttt{ANANAS}$

$S = \texttt{BATMAN AND ANNA SING NANANANA AND EAT BANANAS}$

0 5 10 15 20 25 30 35 40

- 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])

Complexities - Algorithms

- $|P| = m$, $|S| = n$
- (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

Complexities - Data structures

- 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:

Complexities - Data structures

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

	space	time	
[AA02]	$O(n)$	$O(\sum_{i=1}^{m} dist\_occ(P_i) \cdot i)$	
This work	$O(n + \left(\frac{n}{\tau}\right)^k)$	$O(k \cdot \tau \cdot \log\log n)$	$m = k$ fixed
This work	$\Omega(n^{k-k\delta-o(1)})$	$O(n^{\delta})$	$m = k$ fixed

Complexities - Data structures

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

	space	time	
[AA02]	$O(n)$	$O(\sum_{i=1}^{m} dist\_occ(P_i) \cdot i)$	
This work	$O(n + \left(\frac{n}{\tau}\right)^k)$	$O(k \cdot \tau \cdot \log\log n)$	$m = k$ fixed
This work	$\Omega(n^{k-k\delta-o(1)})$	$O(n^{\delta})$	$m = k$ fixed

- $dist\_occ(P_i)$ 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:

	space	time	
[AA02]	$O(n)$	$O(\sum_{i=1}^{m} dist\_occ(P_i) \cdot i)$	
This work	$O(n + \left(\frac{n}{\tau}\right)^k)$	$O(k \cdot \tau \cdot \log\log n)$	$m = k$ fixed
This work	$\Omega(n^{k-k\delta-o(1)})$	$O(n^{\delta})$	$m = k$ fixed

- $dist\_occ(P_i)$ is the number of distinct minimal substrings containing $P[1] \dots P[i]$ as a subsequence
- Conditional lower bound based on hardness of $k-$Set Disjointness

Complexities - Special case $|P| = 2$

- 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 $O(n^{2-\epsilon}\mathrm{poly}(d))$

OV$\rightarrow$ Episode Matching

- build P from B: for $b \in B$, seperate each coordinate by new letter x
  eg: $101 \rightarrow 1x0x1$

OV$\longrightarrow$ Episode Matching

- build P from B: for $b \in B$, seperate each coordinate by new letter x
  eg: $101 \longrightarrow 1x0x1$
- concatenate and separate by new letter y
  eg: $B = \{101, 111, 110\}$,
  $P = 1x0x1y1x1x1y1x1x0$

OV$\to$ Episode Matching

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

OV$\rightarrow$ Episode Matching

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

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

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

OV$\to$ Episode Matching

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

$$0 \longrightarrow 01$$
$$1 \longrightarrow 00$$

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

OV→ Episode Matching

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

$$0 \longrightarrow 01$$
$$1 \longrightarrow 00$$

  separate each coordinate by letter x
  eg: $100 \longrightarrow 00x01x01$
- let $b_1 = 010$, $b_2 = 110$, $a = 100$

OV$\rightarrow$ Episode Matching

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

$$0 \longrightarrow 01$$
$$1 \longrightarrow 00$$

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

- let $b_1 = 010$, $b_2 = 110$, $a = 100$
- b_1 and a are orthogonal

OV$\rightarrow$ 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 $00x01x01$

OV$\longrightarrow$ Episode Matching

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

$$0 \longrightarrow 01$$
$$1 \longrightarrow 00$$

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

- 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$\rightarrow$ 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

a	s(a)	b	p(b)
010	01x00x01	010	0x1x0

- to build S as follows:

OV$\rightarrow$ Episode Matching

a	s(a)	b	p(b)
010	01x00x01	010	0x1x0

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

OV→ Episode Matching

a	s(a)	b	p(b)
010	01x00x01	010	0x1x0

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

OV→ Episode Matching

a	s(a)	b	p(b)
010	$01x00x01$	010	$0x1x0$

- to build S as follows:
- let z be the d-dimensional 0 vector
- $\Rightarrow s(z) = 01x01x\dots x01$
- $S =$ $s(a_1)ys(z)ys(a_2)ys(z)y\dots s(a_n)ys(z)ys(a_1)ys(z)y\dots s(n)$

OV$\to$ Episode Matching

a	s(a)	b	p(b)
010	$01x00x01$	010	$0x1x0$

- to build S as follows:
- let z be the d-dimensional 0 vector
- $\Rightarrow s(z) = 01x01x\dots x01$
- $S =$ $s(a_1)ys(z)ys(a_2)ys(z)y\dots s(a_n)ys(z)ys(a_1)ys(z)y\dots s(n)$
- Length of $S = O(nd)$

OV→ Episode Matching

a	s(a)	b	p(b)
010	$01x00x01$	010	$0x1x0$

- to build S as follows:
- let z be the d-dimensional 0 vector
- $\Rightarrow s(z) = 01x01x \dots x01$
- $S =$ $s(a_1)ys(z)ys(a_2)ys(z)y \dots s(a_n)ys(z)ys(a_1)ys(z)y \dots s(n)$
- Length of $S = O(nd)$
- $|P||S|^{1-\epsilon} = O(n^{2-\epsilon}d^{2-\epsilon})$
  $|P|^{1-\epsilon}|S| = O(n^{2-\epsilon}d^{2-\epsilon})$

OV→ Episode Matching

a	s(a)	b	p(b)	z	s(z)
010	01x00x01	010	0x1x0	000	01x01x01

No orthogonal vectors:

$y \quad s(z) \quad y \quad s(a_{i-1}) \quad y \quad s(z) \quad y \quad s(a_i) \quad y \quad s(z) \quad y \; s(a_{i+1})$

$y \; p(b_{j-1}) \quad y \; p(b_j) \quad y \; p(b_{j+1})$

OV→ Episode Matching

a	s(a)	b	p(b)	z	s(z)
010	01x00x01	010	0x1x0	000	01x01x01

a_i, b_j orthogonal:

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	$p(b_{j+1})$		

OV→ Episode Matching

$$S = \underline{s(a_1)ys(z)ys(a_2)ys(z)y \dots s(a_n)}ys(z)ys(a_1)ys(z)y \dots s(a_n)$$

- $a_i \perp b_j$
- $j < i$: "overflow" to the right
- $j > i$: "overflow" to the left

OV→ Episode Matching

$\ldots p(b_2) y \, p(b_3) y \quad p(b_4) \ldots$

$$S = \underline{s(a_1){\color{red}y}s(z){\color{red}y}s(a_2){\color{red}y}s(z){\color{red}y}\ldots s(a_n)}{\color{red}y}s(z){\color{red}y}s(a_1){\color{red}y}s(z){\color{red}y}\ldots s(a_n)$$

- $a_i \perp b_j$
- $j < i$: “overflow” to the right
- $j > i$: “overflow” to the left

OV→ Episode Matching

$\dots y\, p(b_4)\, y\, p(b_5)\, y\, p(b_6) \dots$

$$S = \underline{s(a_1)ys(z)ys(a_2)ys(z)y\dots s(a_n)}ys(z)ys(a_1)ys(z)y\dots s(a_n)$$

- $a_i \perp b_j$
- $j < i$: "overflow" to the right
- $j > i$: "overflow" to the left

Binary alphabet

- replace x and y by binary gadgets

Space/time trade-off

- $|P| = k$ fixed at preprocessing
- Upper bound: Space: $O(n + \left(\frac{n}{\tau}\right)^k)$, Time: $O(k \cdot \tau \cdot \log\log n)$
  $m = k$
- Conditional lower bound: Space: $\Omega(n^{k - k\delta - o(1)})$, Time: $O(n^{\delta})$

Space/time trade-off, Lower bound

Definition (k-Set Disjointness Problem)
Preprocess m sets S_1, S_2, ..., 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 $\bigcap_{j=1}^{k} S_{i_j} = \emptyset$.

Space/time trade-off, Lower bound

Definition (k-Set Disjointness Problem)
Preprocess m sets S_1, S_2, ..., 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]

Space/time trade-off, Lower bound

Definition (k-Set Disjointness Problem)
Preprocess m sets S_1, S_2, ..., 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 $\tilde{\Omega}\left(N^k / T^k\right)$ space.

Space/time trade-off, Lower bound

$$
\begin{aligned}
S_1 &= \{1, 3, 4\} & \alpha_1 \\
S_2 &= \{2\} & \alpha_2 \\
S_3 &= \{1, 2, 3, 4\} & \alpha_3 \\
S_4 &= \{2, 4\} & \alpha_4 \\
S_5 &= \{1, 3\} & \alpha_5
\end{aligned}
$$

Space/time trade-off, Lower bound

$$
\begin{array}{ll}
S_1 = \{1,3,4\} & \alpha_1 \\
S_2 = \{2\} & \alpha_2 \\
S_3 = \{1,2,3,4\} & \alpha_3 \\
S_4 = \{2,4\} & \alpha_4 \\
S_5 = \{1,3\} & \alpha_5
\end{array}
$$

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

Space/time trade-off, Lower bound

$$\begin{array}{ll} S_1 = \{1,3,4\} & \alpha_1 \\ S_2 = \{2\} & \alpha_2 \\ S_3 = \{1,2,3,4\} & \alpha_3 \\ S_4 = \{2,4\} & \alpha_4 \\ S_5 = \{1,3\} & \alpha_5 \end{array}$$

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

$S_1 \cap S_4 = \emptyset$?

Space/time trade-off, Lower bound

$$\begin{aligned} S_1 &= \{1,3,4\} & \alpha_1 \\ S_2 &= \{2\} & \alpha_2 \\ S_3 &= \{1,2,3,4\} & \alpha_3 \\ S_4 &= \{2,4\} & \alpha_4 \\ S_5 &= \{1,3\} & \alpha_5 \end{aligned}$$

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

$S_1 \cap S_4 = \emptyset$?

$P_1 = \alpha_1\alpha_4$

Space/time trade-off, Lower bound

$$S_1 = \{1,3,4\} \qquad \alpha_1$$
$$S_2 = \{2\} \qquad \alpha_2$$
$$S_3 = \{1,2,3,4\} \qquad \alpha_3$$
$$S_4 = \{2,4\} \qquad \alpha_4$$
$$S_5 = \{1,3\} \qquad \alpha_5$$

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

$S_1 \cap S_4 = \emptyset$?

$P_1 = \alpha_1\alpha_4$

$S_2 \cap S_5 = \emptyset$?

Space/time trade-off, Lower bound

$$\begin{aligned} S_1 &= \{1,3,4\} & \alpha_1 \\ S_2 &= \{2\} & \alpha_2 \\ S_3 &= \{1,2,3,4\} & \alpha_3 \\ S_4 &= \{2,4\} & \alpha_4 \\ S_5 &= \{1,3\} & \alpha_5 \end{aligned}$$

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

$S_1 \cap S_4 = \emptyset$?

$P_1 = \alpha_1\alpha_4$

$S_2 \cap S_5 = \emptyset$?

$P_2 = \alpha_2\alpha_5$

Space/time trade-off, upper bound

- Space$=O(n+\left(\frac{n}{\tau}\right)^k)$, 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 τ 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 τ times *frequent*
- For all $k-$tuples of frequent letters precompute answers

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 τ 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 τ 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 τ 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 τ 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

$P=$ `ANANAS`

$S=$ `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 τ 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

$P=$ ANANAS

$S=$ 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 τ 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

$P =$ ANANAS

$S =$ BATMAN AND ANNA SING NANANANANA 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 τ 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

$P=$ `ANANAS`

$S=$ `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 τ 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

$P =$ `ANANAS`

$S =$ `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 τ 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

$P=$ ANANAS

$S=$ BATMAN AND ANNA SING NANANANANA AND EAT BANANAS

0 5 10 15 20 25 30 35 40

Thank you!

Contact: teresa.anna.steiner@univie.ac.at

References

Alberto Apostolico and Mikhail J. Atallah.
Compact recognizers of episode sequences.
Inf. Comput., 174(2):180–192, 2002.

Philip Bille, Inge Li Gørtz, Max Rishøj Pedersen, and Teresa Anna Steiner.
Gapped indexing for consecutive occurrences.
In *Proc. 32nd CPM*, pages 10:1–10:19, 2021.

Gautam Das, Rudolf Fleischer, Leszek Gasieniec, Dimitrios Gunopulos, and Juha Kärkkäinen.
Episode matching.
In *Proc. 8th CPM*, pages 12–27, 1997.

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