The Fine-Grained Complexity of Episode Matching

Philip Bille ✉ 📷
Technical University of Denmark, Denmark

Inge Li Gørtz ✉ 📷
Technical University of Denmark, Denmark

Shay Mozes ✉ 📷
The Interdisciplinary Center Herzliya, Israel

Teresa Anna Steiner ✉ 📷
Technical University of Denmark, Denmark

Oren Weimann ✉ 📷
University of Haifa, Israel

―――― **Abstract** ――――

Given two strings S and P, the *Episode Matching* problem is to find the shortest substring of S that contains P as a subsequence. The best known upper bound for this problem is $\tilde{O}(nm)$ by Das et al. (1997), where n, m are the lengths of S and P, respectively. Although the problem is well studied and has many applications in data mining, this bound has never been improved. In this paper we show why this is the case by proving that no $O((nm)^{1-\epsilon})$ algorithm (even for binary strings) exists, unless the *Strong Exponential Time Hypothesis (SETH)* is false.

We then consider the indexing version of the problem, where S is preprocessed into a data structure for answering episode matching queries P. We show that for any τ, there is a data structure using $O(n + \left(\frac{n}{\tau}\right)^k)$ space that answers episode matching queries for any P of length k in $O(k \cdot \tau \cdot \log \log n)$ time. We complement this upper bound with an almost matching lower bound, showing that any data structure that answers episode matching queries for patterns of length k in time $O(n^\delta)$, must use $\Omega(n^{k-k\delta-o(1)})$ space, unless the *Strong k-Set Disjointness Conjecture* is false. Finally, for the special case of $k = 2$, we present a faster construction of the data structure using fast min-plus multiplication of bounded integer matrices.

1 Introduction

A string P is a *subsequence* of a string S if P can be obtained by deleting characters from S. Given two strings S and P, the *Episode Matching* problem [15] is to find the shortest substring of S that contains P as a subsequence. In its indexing version, we are given S in advance and we need to preprocess it into a data structure for answering episode matching queries P.

1.1 Episode matching

The Episode Matching problem was introduced by Das et al. [15] in 1997 as a simplified version of the frequent episode discovery problem first studied by Mannila et al. [31]. Das et al. [15]

gave an upper bound of $O(nm/\log n)$, where n is the length of S and m is the length of P. Even though the problem and its variations have been thoroughly studied [6, 11, 13, 14, 29, 30] and have many applications in data mining [6, 7, 22, 26, 31], the original $O(nm/\log n)$ upper bound has never been improved. In Section 3 we show why this is the case, by proving a lower bound conditioned on the *Strong Exponential Time Hypothesis (SETH)* (actually, on the Orthogonal Vectors Hypothesis (OV)). This continues a recent line of research on quadratic lower bounds for string problems conditioned on SETH [1, 3, 8, 9, 12, 16, 17, 19, 27, 33]. Our reduction is very simple and easily extends to binary alphabet and to unbalanced inputs. Our result is summarized by the following theorem.

▶ **Theorem 1.** *For any $\epsilon > 0$, Episode Matching on binary strings of lengths n and $m = n^\alpha$ (for any fixed $\alpha \leq 1$) cannot be solved in $O((mn)^{1-\epsilon})$ time, unless SETH is false.*

Related work. A related problem to episode matching is the *Longest Common Subsequence (LCS)* problem. In that problem, the goal is to find for two strings S and P (of lengths n and m respectively) the longest string that is a subsequence of both S and P. In 2015 Abboud et al. [1] and Bringmann and Künnemann [12] independently proved quadratic lower bounds for LCS conditioned on the SETH. That is, they showed that assuming SETH, there cannot be an $O(m^{2-\epsilon})$ algorithm. However, there exists an $\tilde{O}(n+m^2)$ algorithm for LCS [24]. This suggests that for unbalanced inputs where $m \ll n$, episode matching is harder than LCS.

Further, the episode matching problem can be seen as a special case of the *Approximate String Matching* problem. There, given strings S and P, the goal is to find the substring of S minimizing some distance measure to P. If this distance measure is the number of deleted characters from S, the problem is equivalent to episode matching. Another version of the approximate string matching problem uses *edit distance*. Backurs and Indyk [9] implicitly give a quadratic lower bound conditioned on SETH for that version as a stepping stone for achieving a lower bound for edit distance, using an alphabet of size seven. This bound does not however directly translate to our problem, and uses a more complicated construction and a larger alphabet.

To avoid misunderstandings, we note that in the paper by Mannila et al. [31], an *episode* was more generally defined as a collection of events that occur together. The string formulation by Das et al. [15] is a simplification of this original definition, thus, the terms *episode* and *episode matching* have been used to name different concepts in the literature, see e.g. [4, 21, 23, 32, 34]. We only consider the string formulation by Das et al. [15].

1.2   Episode Matching Indexing

Limited by the above lower bound for episode matching, a natural question to ask is what bounds we can get if we allow preprocessing. Alas, in terms of time complexity, preprocessing does not help. Namely, our reduction is such that each of the two sets in the Orthogonal Vectors instance (see Definition 6) is encoded *independently* into one of the two strings in the episode matching instance. This implies (see e.g. [18]) that episode matching cannot be solved in $O((mn)^{1-\epsilon})$ time even if we are allowed polynomial time to preprocess one of the two strings in advance. In other words, polynomial time preprocessing is not enough to guarantee subquadratic time queries. In fact, this holds even when one of the strings is fixed (cf. [2, Section 1.1.2]). We therefore focus on time-space tradeoffs.

Formally, given a string S and an integer k, the *episode matching indexing* problem asks to construct a compact data structure that can quickly report the episode matching of any pattern P of length k (i.e. compute the length of the shortest substring of S that

contains P as a subsequence). Apostolico and Atallah [6] gave a linear (optimal) space data structure whose query time may be prohibitive (depending on the number of distinct minimal substrings containing a prefix of the pattern as a subsequence). In Section 4.1 we present a time-space tradeoff with faster query time:

▶ **Theorem 2.** *For any τ, there is a data structure using space $O(n + \left(\frac{n}{\tau}\right)^k)$ that answers Episode Matching queries in time $O(k \cdot \tau \cdot \log \log n)$.*

In Section 4.2 we give the following almost matching lower bound, conditioned on the k-Set Disjointness Conjecture:

▶ **Theorem 3.** *For any $\delta \in [0, 1/2]$, a data structure that answers Episode Matching queries in time $O(n^\delta)$ must use $\Omega(n^{k-k\delta-o(1)})$ space, unless the Strong k-Set Disjointness Conjecture is false.*

Finally, in Section 4.3 we consider the decision version of the case $k = 2$. That is, we are given some threshold t, and a query (a, b) need only report whether S contains a substring of length at most t that starts with the letter a and ends with the letter b. In this setting, there is a simple $O(1)$-query time $O(\sigma^2)$-space data structure for episode matching (where σ is the size of the alphabet): precompute and store the $\sigma \times \sigma$ matrix D with the answers to every possible query. Naively, we can compute D in time $\min\{\tilde{O}(n\sigma + \sigma^2), O(nt + \sigma^2)\}$. We show that for various values of σ and t, we can compute the matrix D faster by using fast min-plus matrix multiplication of bounded integer matrices [37]:

▶ **Theorem 4.** *For a given threshold t, we can compute the matrix D in time*
$O(\sigma^{1/2+\omega/2} \left(\frac{n}{t}\right)^{1/2} \sqrt{n})$ *if $\sigma < \frac{n}{t}$, and*
$O(\sigma^2 \left(\frac{n}{t}\right)^{\omega/2-1} \sqrt{n})$ *if $\sigma \geq \frac{n}{t}$.*

In particular, using the current matrix multiplication exponent $\omega < 2.4$ [5], if $\sigma < \frac{n}{t}$ then we get $O(\sigma^{1.7} n t^{-1/2})$ (which is smaller than $\min(n\sigma, nt)$ for any $t > \sigma^{1.4}$), and if $\sigma \geq \frac{n}{t}$ we get $O(\sigma^{2.2}\sqrt{n})$ (which is smaller than $O(n\sigma)$ for any $\sigma < n^{0.416}$).

2 Preliminaries

In this section we will review some basic string notation and important hypotheses.

A string S of length n is a sequence $S[0]\cdots S[n-1]$ of n characters drawn from an alphabet Σ of size $|\Sigma| = \sigma$. A string $S[i\cdots j] = S[i]\cdots S[j]$ for $0 \leq i < j < n$ is called a *substring* of S. For two strings X and Y we denote their *concatenation* as $X \cdot Y$ or XY.

The Strong Exponential Time Hypothesis is a popular conjecture about the hardness of the k-SAT problem, postulated by Impagliazzo and Paturi [25]. The k-SAT problem is to decide whether there exists a satisfying assignment for a Boolean formula on n variables and clauses of width at most k.

▶ **Conjecture 5** (The Strong Exponential Time Hypothesis). *There is no $\epsilon > 0$ for which k-SAT can be solved in time $O(2^{(1-\epsilon)n})$ for all $k \geq 3$.*

Instead of reducing directly from k-SAT, we reduce from the Orthogonal Vectors problem, and use two conjectures about the hardness of Orthogonal Vectors which are implied by SETH for $d = \omega(\log n)$.

▶ **Definition 6** (Orthogonal Vectors Problem (OV)). *Given two sets $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ of d-dimensional binary vectors, decide whether there is an orthogonal pair of vectors $a_i \in A$ and $b_j \in B$.*

The two conjectures consider the cases of equal set size and unbalances set size, respectively. They roughly state that any algorithm solving OV that has a polynomial dependency on the dimension (denoted poly(d)), cannot achieve a significantly better asymptotic running time than the product of the two set sizes.

▶ **Conjecture 7** (Orthogonal Vectors Hypothesis (OVH)). *For $|A| = |B| = n$, there is no $\epsilon > 0$ for which OV can be solved in time $O(n^{2-\epsilon}\text{poly}(d))$.*

▶ **Conjecture 8** (Unbalanced Orthogonal Vectors Hypothesis (UOVH)). *Let $0 < \alpha \leq 1$, $|A| = n$ and $|B| = m$. There is no $\epsilon > 0$ for which OV restricted to $m = \Theta(n^\alpha)$ and $d = n^{o(1)}$ can be solved in time $O((nm)^{1-\epsilon})$.*

It is known that SETH implies both OVH and UOVH [12, 36]. Finally, we consider the following conjecture of Goldstein et al. [20] on the k-Set Disjointness problem:

▶ **Definition 9** (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 $\bigcap_{j=1}^{k} S_{i_j} = \emptyset$.*

▶ **Conjecture 10** (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.*

3 Episode Matching

In this section we prove Theorem 1. We first prove it for an alphabet of size four, and then for a binary alphabet.

3.1 Alphabet of Size Four

We show how to reduce an instance of OV to Episode Matching with alphabet $\{0, 1, x, \$\}$. Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ be two sets of vectors in $\{0, 1\}^d$. Without loss of generality, we assume $m \leq n$. We show how to construct a string P of length $2dm + 1$ and a string S of length $3d(4n + 1) + 1$ such that there is a pair of orthogonal vectors $a_i \in A$ and $b_j \in B$ if and only if there is a substring of S of length at most $3d(2m - 1) + 1$ that contains P as a subsequence.

Constructing P. We construct P from the set B in the following way. For every $b \in B$, we define $p(b)$ as the string of length $2d - 1$ obtained by inserting an x symbol between each consecutive entries of b. That is,

$$p(b) = b[0] \ x \ b[1] \ x \ \cdots x \ b[d]$$

Then, P is a string of length $2dm + 1$ defined as the concatenation:

$$P = \$ \ p(b_1) \ \$ \ p(b_2) \ \$ \cdots \$ \ p(b_m) \ \$$$

Constructing S. We construct S from the set A. For a vector a in $\{0, 1\}^d$, we define for each entry in a a string of length 2, called a *coordinate gadget*:

$$s(a[i]) = 01 \text{ if } a[i] = 0,$$

$$s(a[i]) = 00 \text{ if } a[i] = 1.$$

Then, $s(a)$ is a string of length $3d - 1$ defined as the concatenation:

$$s(a) = s(a[0]) \ x \ s(a[1]) \ x \cdots x \ s(a[d])$$

For example, the vector $a = 10001$ defines the string $s(a) = 00x01x01x01x00$. Let z be the $d-$dimensional zero vector, and so $s(z) = 01x01x \ldots x01$. Then, S consists of \$ $s(z)$ \$ followed by two copies of the concatenation of $s(a_i)$ \$ $s(z)$ \$ for $1 \leq i \leq n$. That is:

$$S = \$s(z)\$s(a_1)\$s(z)\$s(a_2)\$ \cdots \$s(z)\$s(a_n)\$s(z)\$s(a_1)\$s(z)\$s(a_2)\$ \cdots \$s(z)\$s(a_n)\$s(z)\$$$

We call a substring of S of the form "$s(a)$ \$" or "$s(z)$ \$" a *block*. A block has length $3d$. The string S contains $2n$ blocks for the elements of A, plus $2n + 1$ blocks for the separating $s(z)$, thus $4n + 1$ blocks in total. There is an extra \$ at the start of the string. Thus, $|S| = 3d(4n + 1) + 1 = \Theta(dn)$.

Correctness. For two strings Y and X, an *alignment L of Y in X* is a sequence $0 \leq j_0 < \cdots < j_{|Y|-1} \leq |X| - 1$ such that $Y[\ell] = X[j_\ell]$ for every $0 \leq \ell \leq |Y| - 1$. We say $Y[\ell]$ is *aligned* to $X[j_\ell]$ and that L *spans* the substring $X[j_0 \cdots j_{|Y|-1}]$. Clearly, Y is a subsequence of X if and only if there exists an alignment of Y in X.

The following Lemma establishes the translation of orthogonality in our reduction.

▶ **Lemma 11.** *For two vectors a and b of dimension d, the string $p(b)$ is a subsequence of $s(a)$ if and only if b and a are orthogonal.*

Proof. If a and b are orthogonal, then for every $b[i] = 1$ we have that $a[i] = 0$ and therefore $s(a[i]) = 01$. Thus we can align $b[i]$ to the 1 in $s(a[i])$. If $b[i] = 0$, we can align $b[i]$ to the first character in $s(a[i])$. We can therefore define an alignment where we align every $b[i]$ to one of the characters in $s(a[i])$, and align the ith x in $p(b)$ to the ith x in $s(a)$. Therefore $p(b)$ is a subsequence of $s(a)$.

On the other hand, if $p(b)$ is a subsequence of $s(a)$, then since $p(b)$ and $s(a)$ both contain exactly $d - 1$ x's, any alignment of $p(b)$ in $s(a)$ *must* align the ith x in $p(b)$ to the ith x in $s(a)$. Thus, each $b[i]$ must be aligned to a character in $s(a[i])$. If $b[i] = 1$, we can align it with a character in $s(a[i])$ only if $a[i] = 0$. Thus, if $p(b)$ is a subsequence, then for every $b[i] = 1$, we must have $a[i] = 0$. Therefore a and b are orthogonal. ◄

Lemma 11 implies that any $p(b)$ is a subsequence of $s(z)$. This immediately yields a substring of S of length $3d(2m - 1) + 1$ which contains P as a subsequence: If we align P with m consecutive occurrences of $s(z)$, the resulting substring contains a \$ at the beginning, m blocks of the form "$s(z)$ \$" and $m - 1$ blocks of the form "$s(a)$ \$". Next, we show that if there is no pair of orthogonal vectors, then there is no shorter substring of S that contains P as a subsequence.

▷ **Claim 12.** If there exist no $a \in A$ and $b \in B$ which are orthogonal, then there exists no substring of S of length $< 3d(2m - 1) + 1$ which contains P as a subsequence.

Proof. Consider an alignment L of P in S. If for every $b \in B$ the substring $p(b)$ is aligned to a string of the form $s(z)$, then L spans a string of length at least $3d(2m - 1) + 1$. Consider now the case where there exists a $b \in B$ such that $p(b)$ is not fully aligned to a string of the form $s(z)$. By Lemma 11, since there is no $a \in A$ such that a and b are orthogonal, there is no $s(a)$ such that $p(b)$ is fully aligned within $s(a)$. Since no $s(a)$ or $s(z)$ contain a \$, this means that the alignment of the string \$ $p(b)$ \$ spans a string containing either a substring of the form \$ $s(a)$ \$ $s(z)$ \$ or a substring of the form \$ $s(z)$ \$ $s(a)$ \$. Then the alignment L'

$$\cdots \$ \quad s(z) \quad \$ \; s(a_{i-1}) \; \$ \quad s(z) \quad \$ \; s(a_i) \; \$ \quad s(z) \quad \$ \; s(a_{i+1}) \; \$ \quad s(z) \quad \$ \cdots$$
$$\cdots \$ \; p(b_{j-2}) \qquad\qquad \$ \; p(b_{j-1}) \; \$ \quad p(b_j) \; \$ \; p(b_{j+1}) \qquad\qquad \$ \; p(b_{j+2}) \; \$ \cdots$$

Figure 1 The alignment of P and S as described in the proof of Claim 13.

defined by aligning $\$ \, p(b) \, \$$ to the $\$ \, s(z) \, \$$ in that substring, and everything else as in L, spans a substring in S no longer than the substring spanned by L. Repeating this for each $p(b)$ that is not aligned to $s(z)$ gives an alignment where every $p(b)$ is aligned to some $s(z)$, and which spans a substring in S no longer than the substring spanned by L. Since this substring contains at least m copies of $\$ \, s(z) \, \$$, it has length at least $3d(2m - 1) + 1$. ◄

Next, we show how to align P in S to yield a shorter substring, if there does exist an orthogonal pair.

▷ **Claim 13.** If there exist $a \in A$ and $b \in B$ which are orthogonal, then there is a substring of S of length $< 3d(2m - 1) + 1$ that contains P as a subsequence.

Proof. Assume a_i and b_j are orthogonal and $j \leq i$. We align $p(b_j)$ to the first copy of $s(a_i)$, and align $p(b_{j+1}) \cdots p(b_m)$ to the next $m - j$ copies of $s(z)$ to the right of $s(a_i)$, and $p(b_1) \cdots (b_{j-1})$ to the previous $j - 1$ copies of $s(z)$ to the left of $s(a_i)$. See Figure 1. This is possible since $j \leq i$. Let T denote the resulting substring of S.

- Case 1: $j = 1$ (or $j = m$). In this case, the substring of T spans $2m - 2$ blocks: It starts (ends) with "$\$ \, s(a_i) \, \$$", and then includes the $m - 1$ following (preceeding) $s(z)$ blocks. Between any two $s(z)$ blocks, there is another block corresponding to some $a_\ell \in A$. Thus in total, the length of T is $3d(2m - 2) + 1 < 3d(2m - 1) + 1$.
- Case 2: $1 < j < m$. The substring T starts and ends with $\$ \, s(z) \, \$$, and we align to $m - 1$ of the $s(z)$ blocks and to $s(a_i)$ which is somewhere inbetween these. Thus, T includes $m - 1 + m - 2 = 2m - 3$ blocks, and the length of T is $3d(2m - 3) + 1 < 3d(2m - 1) + 1$.

If $j > i$, we align $p(b_j)$ to the *second* copy of $s(a_i)$, and again align $p(b_{j+1}) \cdots p(b_m)$ to the next $m - j$ copies of $s(z)$ to the right of $s(a_i)$, and $p(b_1) \cdots p(b_{j-1})$ to the preceding $j - 1$ copies of $s(z)$ to the left of $s(a_i)$. This is possible since $m - j \leq n - i$. The rest follows analogously. ◄

Analysis. When $m = \Theta(n)$, we have that $|P| = \Theta(|S|) = \Theta(nd)$, which means that an $O((|S||P|)^{1-\epsilon})$ algorithm for Episode Matching implies an $O(n^{2-2\epsilon} d^{2-2\epsilon})$ algorithm for OV, contradicting OVH.

When $m = \Theta(n^\alpha)$ for some $0 < \alpha < 1$, and $d = n^{o(1)}$, the length of P is $2dm + 1 = mn^{o(1)}$, and the length of S is $3d(4n + 1) + 1 = n^{1+o(1)}$. This means that an $O((|S||P|)^{1-\epsilon})$ algorithm for Episode Matching implies an $(nm)^{1-\epsilon} n^{o(1)} = O((nm)^{1-\epsilon'})$ algorithm for OV for any $\epsilon' < \epsilon$, contradicting UOVH.

This proves Theorem 1 for alphabet size at least 4.

3.2 Binary Alphabet

We now prove Theorem 1 for a binary alphabet. We will use almost the same reduction as before, but replace x and $\$$ with binary gadgets.

Inner separator gadget. Instead of the separating symbol x, we define an *inner separator gadget* of the form 0^d. The strings $p(b)$ and $s(a)$ are defined as before, except that each x is

substituted with the inner separator gadget 0^d, and we add an extra inner separator gadget at the beginning and end of the string. That is,

$$p(b) = 0^d \ b[1] \ 0^d \ b[2] \ 0^d \cdots 0^d \ b[d] \ 0^d$$

and

$$s(a) = 0^d \ s(a[1]) \ 0^d \ s(a[2]) \ 0^d \cdots 0^d \ s(a[d]) \ 0^d \ .$$

Outer separator gadget. Instead of the separating symbol \$, we define an *outer separator gadget* of the form 1^{d+1}. The strings P and S are defined as before, with the new versions of $p(b)$, $s(a)$ and $s(z)$, and every \$ substituted with the outer separator gadget 1^{d+1}.

For any $b \in B$, the length of the string $p(b)$ is $(d+2)d$, since it contains $d+1$ inner separator gadgets of length d. The string P consists of $m+1$ outer separator gadgets, each of length $d+1$, and m substrings $p(b)$ each of length $(d+2)d$. The length of P is therefore $(d+2)dm + (m+1)(d+1) = \Theta(md^2)$.

For any $a \in A$, the length of the string $s(a)$ is $(d+3)d$. Analogously to before, we define a *block* as a substring of the form $s(a) \ 1^{d+1}$. The length of a block is $(d+3)d+(d+1) = d^2+4d+1$. The string S consists of $4n+1$ blocks (as before), each of length d^2+4d+1 and an extra outer separator gadget at the beginning. The length of S is therefore $(d^2+4d+1)(4n+1)+d+1 = \Theta(nd^2)$.

Now, if we align every $p(b)$ to a copy of $s(z)$, as in the proof in Section 3.1, we need $2m-1$ blocks. Thus, we get a substring of length $w = (d^2+4d+1)(2m-1)+d+1$. Our reduction will again depend on the fact that if there are no orthogonal vectors, we cannot do much better than that. Namely, we next prove that if there is no pair of orthogonal vectors then there is no substring of S of length $< w - 2d$ that contains P as a subsequence (Claim 15), and that if there is a pair of orthogonal vectors then there is a substring of S of length $\leq w - (d^2+4d+1)$ that contains P as a subsequence (Claim 16).

Algorithm. We run the assumed blackbox algorithm for Episode Matching on S and P. If the algorithm outputs a substring of length at least $w - 2d$, we conclude that there are no orthogonal vectors. If it outputs some string of a shorter length, then there are.

Correctness. The correctness proof follows along the same lines as in the alphabet four case. We begin by proving an equivalent version of Lemma 11:

▶ **Lemma 14.** *For two vectors a and b of dimension d, the string $p(b)$ is a subsequence of $s(a)$ if and only if b and a are orthogonal.*

Proof. If a and b are orthogonal, then by the same arguments as before, $p(b)$ is a subsequence of $s(a)$. Otherwise, a and b are not orthogonal, so there exists an i such that $b[i] = a[i] = 1$. Assume for the sake of contradiction that $p(b)$ is a subsequence of $s(a)$. Then we must align the 1 corresponding to $b[i]$ with some coordinate gadget $s(a[j]) = 01$ and $j \neq i$. There are two cases depending on whether $i > j$ or $j > i$.

- Case 1: $i > j$. Note that to the left of the 1 corresponding to $b[i]$, there are $(i-1) + di$ characters in $p(b)$, and to the left of the 1 in $s(a[j]) = 01$ there are $2(j-1) + dj + 1$ characters in $s(a)$: namely $j-1$ coordinate gadgets, j inner separator gadgets, and the 0 in $s(a[j])$. But this means that the prefix of $p(b)$ up to $b[i]$ is longer than the prefix of $s(a)$ up to the 1 in $s(a[j])$, since

$$(i-1) + id \ \geq \ j + (j+1)d \ = \ j + d + jd \ > \ 2(j-1) + jd + 1,$$

where the last inequality holds because $j < d$. Thus we cannot have aligned the prefix of $p(b)$ up to $b[i]$ to the prefix of $s(a)$ up to the 1 in $s(a[j])$, contradicting that $p(b)$ is a subsequence of $s(a)$.

- Case 2: $j > i$. Note that to the right of the 1 corresponding to $b[i]$, there are $d - i + (d - i + 1)d = (d - i)(d + 1) + d$ characters in $p(b)$, and to the right of the 1 in $s(a[j]) = 01$ there are $2(d - j) + (d - j + 1)d = (d - j)(d + 2) + d$ characters in $s(a)$. But this means that the suffix of $p(b)$ to the right of $b[i]$ is longer than the suffix of $s(a)$ to the right of $s(a[j])$, since

$$(d - i)(d + 1) + d \geq (d - j + 1)(d + 1) + d = (d - j)(d + 1) + 2d + 1 > (d - j)(d + 2) + d.$$

This means that we cannot have aligned the suffix of $p(b)$ to the right of $b[i]$ to the suffix of $s(a)$ to the right of $s(a[j])$, contradicting that $p(b)$ is a subsequence of $s(a)$. ◄

Next we prove that if there are no vectors $a \in A$ and $b \in B$ which are orthogonal, then we cannot do much better than aligning $p(b_1), \ldots, p(b_m)$ to m consecutive copies of $s(z)$. Recall that we defined the length of that alignment as w.

▷ **Claim 15.** If there exist no $a \in A$ and $b \in B$ which are orthogonal, then there exists no substring of S of length $< w - 2d$ that contains P as a subsequence.

Proof. Assume an alignment such that $p(b_i)$ is not aligned to $s(z)$. Since there are no orthogonal vectors, there is no $s(a)$ such that $p(b_i)$ is aligned in $s(a)$. Further, since $p(b_i)$ starts and ends with a 0, its alignment cannot start or end within an outer separator gadget. Thus, the alignment of $p(b_i)$ spans a string either containing

1. a non-empty suffix of some $s(a_j)$, followed by 1^{d+1}, followed by a non-empty prefix of $s(z)$ or
2. a non-empty suffix of $s(z)$, followed by 1^{d+1}, followed by a non-empty prefix of some $s(a_j)$.

Consider case 1. Since $s(z)$ only contains d 1s, at least one character in the outer separator gadget following $p(b_i)$ cannot be aligned in $s(z)$. Thus, at least one 1 cannot be aligned before the 1^{d+1} following $s(z)$. If $i < m$, then the inner separator gadget at the beginning of $p(b_{i+1})$ cannot be aligned before the beginning of a new block. Thus, the alignment defined by aligning $p(b_i) \; 1^{d+1}$ to $s(z) \; 1^{d+1}$ spans a string no longer than the original alignment. If $i = m$, the same alignment spans a string at most d longer than the original alignment. Analogously, for case 2, we align $1^{d+1} \; p(b_i)$ to $1^{d+1} \; s(z)$. The alignment spans a string of the same length as the original alignment, or at most d longer if $i = 1$. Repeating this step for any $p(b_i)$ which is not aligned to some $s(z)$, we get an alignment where every $p(b_i)$ is aligned to some $s(z)$, and which spans a string at most $2d$ longer than the original alignment. This concludes the proof. ◄

▷ **Claim 16.** If there exist a and b which are orthogonal, then there is a substring of S of length $\leq w - (d^2 + 4d + 1)$ (i.e. it is shorter than w by at least a block's length) that contains P as a subsequence.

Proof. The proof is analogous to that of Claim 13. ◄

Analysis. The length of S is now $\Theta(nd^2)$, and the length of P is $\Theta(md^2)$. The contradictions to OVH and UOVH are constructed analogously to the alphabet four case. This proves Theorem 1 restricted to binary alphabet.

4   Episode Matching Indexing

In this section we consider the indexing version of episode matching: Given the string S and an integer k, construct a data structure that can quickly report the episode matching of any pattern P of length k.

4.1   Upper bound

We now prove Theorem 2. We call letters that appear more than τ times *frequent* letters. Note that there are at most $\frac{n}{\tau}$ frequent letters. For every k-tuple of frequent letters, we precompute and store the answer in a table. The size of this table is therefore $\left(\frac{n}{\tau}\right)^k$. Additionally, for each letter in the alphabet, we store a predecessor data structure containing all positions in S where the letter appears. Using a linear-space predecessor data structure such as y-fast-trie [35], this requires an additional $O(n)$ space (every position in S appears in exactly one predecessor structure) and answers predecessor/successor queries in $O(\log \log n)$ time.

To answer a query, given a pattern P of length k, if every letter in P is frequent, we simply return the precomputed answer from the table. Otherwise, suppose $P[j]$ is a non-frequent letter. For each position i s.t $S[i] = P[j]$, we find the minimal substring of S that contains P and aligns $P[j]$ to $S[i]$ (eventually we return the smallest substring found). To do this, we start from location i and use successor queries to find $P[j+1], P[j+2], \ldots, P[m-1]$ and predecessor queries to find $P[j-1], P[j-2], \ldots, P[0]$. Since there are at most τ positions in S that contain $P[j]$, this takes overall $O(\tau \cdot k \log \log n)$ time.

4.2   Lower bound

We now prove Theorem 3. The proof is similar to the ones in [28] and [10].

Recall that in the k-Set Disjointness problem, given sets $S_1, \ldots, S_m$ of total size N over a universe U, we want a data structure that given $(i_1, \ldots, i_k)$ reports whether $\bigcap_{j=1}^{k} S_{i_j} = \emptyset$. As in [10], we can reduce k-Set Disjointness to $O(\log N)$ instances of k-Set Disjointness (each of size $O(N)$) with the property that every element in U appears in the same number of sets (call this number f). We next show a reduction from such an instance to episode matching indexing.

Define for each set S_i a unique letter α_i. For each distinct element $e \in U$ define a block consisting of the letters α_i corresponding to the sets S_i that contain e, sorted by i. We then append all such blocks in an arbitrary order and separate each two blocks by an extra block of the form $\$^f$ (where $\$$ is an extra letter not corresponding to any set). The resulting string S is of length $2N - f = O(N)$. We preprocess S into a data structure for episode matching. To answer a query $(i_1, \ldots, i_k)$ (we assume w.l.o.g that $i_1 < i_2 < \cdots < i_k$), we query the data structure for $P = \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}$. If the output is larger than f, we answer that the sets are disjoint, otherwise we answer that they are not. The correctness is simple: If there is an element e that appears in all sets $S_{i_1}, \ldots, S_{i_k}$, then e's block contains $\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_k}$ (and since we sorted the elements, they will appear in the right order). If on the other hand the sets are disjoint, then there is no block containing all letters, so the smallest substring containing all letters must include at least one $\$^f$ block (and is thus longer than f).

The total space for all the $O(\log N)$ instances is therefore $O(\log N \cdot s_{episode}(N))$ and the runtime is $O(\log N \cdot t_{episode}(N))$, where $s_{episode}(n)$ and $t_{episode}(n)$ are the space and query time for a data structure for episode Matching indexing. To prove Theorem 3, assume for contradiction that $t_{episode}(n) = O(n^\delta)$ and $s_{episode}(n) = O(n^{k-k\delta-\epsilon})$. Then the space for solving k-Set Disjointness is $O(N^{k-k\delta-\epsilon}\log N) = O(N^{k-k\delta-\epsilon'})$, for any $0 < \epsilon' < \epsilon$, and the

time is $O(N^\delta \log N) = O(N^{\delta+\epsilon''})$, for arbitrarily small $\epsilon'' > 0$. Setting $\epsilon'' < \epsilon'/k$, we obtain a contradiction to Conjecture 10.

4.3 The special case of $k = 2$

We now prove Theorem 4. Recall that, given the string S, our goal is to compute the binary matrix D where $D[a, b] = 1$ if and only if S contains a substring of length at most t that starts with the letter a and ends with the letter b. Naively, we can compute D in time $\min\{\tilde{O}(n\sigma + \sigma^2), O(nt + \sigma^2)\}$. To get $\tilde{O}(n\sigma + \sigma^2)$, we construct a predecessor data structure for each letter, containing the occurrences of this letter in S. Then, for each position in S we find the succeeding occurrence of each letter in the alphabet, using the corresponding predecessor data structure. Whenever we find a pair of letters (a, b) at distance at most t, we set $D[a, b]$ to 1. To get $O(nt + \sigma^2)$, for each position in S we scan t entries forward and update D as we go. In the remainder of this section, we will show how to compute D faster using min-plus multiplication of bounded integer matrices:

▶ **Lemma 17** ([37])**.** *Given two $n \times n$ matrices A and B, where A has entries in $\{-M, \dots, M\} \cup \{\infty\}$ and B is arbitrary, we can compute their min-plus product $C = A \odot B$ (defined as $C[i, j] = \min_k(A[i, k] + B[k, j])$) in $\tilde{O}(\sqrt{M}n^{(3+\omega)/2})$ time.*

First, we divide the string S into blocks $B_1, B_2, \dots, B_{n/t}$, each of length t (except maybe the last). Let $\delta_{\text{first}}(a, j)$ (resp. $\delta_{\text{last}}(a, j)$) be the distance from the beginning (resp. end) of the jth block to the first (resp. last) a in that block, and ∞ if the block has no a. Note that ab is a subsequence of a length t substring of S if and only if one of the following two conditions holds:

Condition 1: There is a j such that $\delta_{\text{first}}(a, j) + \delta_{\text{last}}(b, j) \leq t$. To check this condition, we check if $(M_1 \odot M_2)[a, b] \leq t$ where M_1 is the $\sigma \times n/t$ matrix with $M_1[a, j] = \delta_{\text{first}}(a, j)$, and M_2 is the $n/t \times \sigma$ matrix with $M_2[j, b] = \delta_{\text{last}}(b, j)$.

Condition 2: There is a j such that $\delta_{\text{last}}(a, j) + \delta_{\text{first}}(b, j + 1) \leq t$. To check this condition, we check if $(M_3 \odot M_4)[a, b] \leq t$ where M_3 is the $\sigma \times (n/t - 1)$ matrix with $M_3[a, j] = \delta_{\text{last}}(a, j)$, and M_4 is the $(n/t - 1) \times \sigma$ matrix with $M_4[j, b] = \delta_{\text{first}}(b, j + 1)$.

By the above discussion, it only remains to compute the products $M_1 \odot M_2$ and $M_3 \odot M_4$ (as then each entry of D can be found in constant time). We focus on $M_1 \odot M_2$ (the other is symmetric). Observe that the entries of M_1 and M_2 are integers bounded by t. Therefore, we can hope to use Lemma 17 (with $M = t$) to multiply them. However, in Lemma 17 the matrices are square while our matrices are rectangular. To deal with this, we break the matrices into rectangular submatrices according to the value of σ.

- If $\sigma < n/t$. We split M_1 into $M_{1,1} \dots, M_{1,\frac{n}{t\sigma}}$, each consisting of σ consecutive columns. We split M_2 into $M_{2,1} \dots, M_{2,\frac{n}{t\sigma}}$, each consisting of σ consecutive rows. Note that $(M_1 \odot M_2)[a, b] = \min_{k=1,\dots,\frac{n}{t\sigma}}(M_{1,k} \odot M_{2,k})[a, b]$. We can thus compute $M_1 \odot M_2$ by computing $\frac{n}{t\sigma}$ min-plus products of $\sigma \times \sigma$ matrices, and setting each entry in $M_1 \odot M_2$ to be the minimum of the corresponding entries in the $\frac{n}{t\sigma}$ output matrices. Using Lemma 17, this takes time $O(\frac{n}{t\sigma}(\sigma^2 + \sigma^{(3+\omega)/2}\sqrt{t})) = O(\sigma^{1/2+\omega/2}\left(\frac{n}{t}\right)^{1/2}\sqrt{n})$.
- If $\sigma \geq n/t$. We split M_1 into $M_{1,1} \dots, M_{1,\frac{t\sigma}{n}}$, each consisting of n/t consecutive rows. We split M_2 into $M_{2,1} \dots, M_{2,\frac{t\sigma}{n}}$, each consisting of n/t consecutive columns. We compute $M_1 \odot M_2$ by computing $\left(\frac{t\sigma}{n}\right)^2$ min-plus products of $n/t \times n/t$ matrices $M_{1,i} \odot M_{2,j}$ for every i, j. Using Lemma 17, this takes time $O(\left(\frac{t\sigma}{n}\right)^2 \cdot \left(\frac{n}{t}\right)^{(3+\omega)/2}\sqrt{t}) = O(\sigma^2\left(\frac{n}{t}\right)^{\omega/2-1}\sqrt{n})$.

This proves Theorem 4.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proc. 56th FOCS*, pages 59–78, 2015.

2 Amir Abboud and Virginia Vassilevska Williams. Fine-grained hardness for edit distance to a fixed sequence. In *Proc. 48th ICALP*, volume 198, pages 7:1–7:14, 2021.

3 Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Proc. 41st ICALP*, pages 39–51, 2014.

4 Avinash Achar, A. Ibrahim, and P. S. Sastry. Pattern-growth based frequent serial episode discovery. *Data Knowl. Eng.*, 87:91–108, 2013.

5 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In Dániel Marx, editor, *Proc. 32nd SODA*, pages 522–539, 2021.

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

7 Mikhail J. Atallah, Robert Gwadera, and Wojciech Szpankowski. Detection of significant sets of episodes in event sequences. In *Proc. 4th ICDM*, pages 3–10, 2004.

8 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *Proc. 57th FOCS*, pages 457–466, 2016.

9 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018.

10 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.

11 Luc Boasson, Patrick Cégielski, Irène Guessarian, and Yuri V. Matiyasevich. Window-accumulated subsequence matching problem is linear. *Ann. Pure Appl. Log.*, 113(1-3):59–80, 2001.

12 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proc. 56th FOCS*, pages 79–97, 2015.

13 Patrick Cégielski, Irène Guessarian, and Yuri V. Matiyasevich. Multiple serial episodes matching. *Inf. Process. Lett.*, 98(6):211–218, 2006.

14 Maxime Crochemore, Costas S. Iliopoulos, Christos Makris, Wojciech Rytter, Athanasios K. Tsakalidis, and T. Tsichlas. Approximate string matching with gaps. *Nord. J. Comput.*, 9(1):54–65, 2002.

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

16 Lech Duraj, Marvin Künnemann, and Adam Polak. Tight conditional lower bounds for longest common increasing subsequence. *Algorithmica*, 81(10):3968–3992, 2019.

17 Massimo Equi, Roberto Grossi, Veli Mäkinen, and Alexandru I. Tomescu. On the complexity of string matching for graphs. In *Proc. 46th ICALP*, pages 55:1–55:15, 2019.

18 Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. In *Proc. 27th SOFSEM*, volume 12607, pages 608–622, 2021.

19 Daniel Gibney. An efficient elastic-degenerate text index? not likely. In *Proc. 27th SPIRE*, pages 76–88, 2020.

20 Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *Proc. 15th WADS*, pages 421–436, 2017.

21 Robert Gwadera, Mikhail J. Atallah, and Wojciech Szpankowski. Markov models for identification of significant episodes. In *Proc. 5th SDM*, pages 404–414, 2005.

22 Robert Gwadera, Mikhail J. Atallah, and Wojciech Szpankowski. Reliable detection of episodes in event sequences. *Knowl. Inf. Syst.*, 7(4):415–437, 2005.

23 Masahiro Hirao, Shunsuke Inenaga, Ayumi Shinohara, Masayuki Takeda, and Setsuo Arikawa. A practical algorithm to find the best episode patterns. In *Proc. 4th DS*, pages 435–440, 2001.

24   Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *J. ACM*, 24(4):664–675, 1977.

25   Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

26   Mika Klemettinen, Heikki Mannila, and Hannu Toivonen. Rule discovery in telecommunication alarm data. *J. Netw. Syst. Manag.*, 7(4):395–423, 1999.

27   Tomasz Kociumaka, Jakub Radoszewski, and Tatiana Starikovskaya. Longest common substring with approximately k mismatches. *Algorithmica*, 81(6):2633–2652, 2019.

28   Tsvi Kopelowitz and Robert Krauthgamer. Color-distance oracles and snippets. In *Proc. 27th CPM*, pages 24:1–24:10, 2016.

29   Josué Kuri, Gonzalo Navarro, and Ludovic Mé. Fast multipattern search algorithms for intrusion detection. *Fundam. Informaticae*, 56(1-2):23–49, 2003.

30   Veli Mäkinen, Gonzalo Navarro, and Esko Ukkonen. Transposition invariant string matching. *J. Algorithms*, 56(2):124–153, 2005.

31   Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, 1997.

32   Elżbieta Nowicka and Marcin Zawada. On the complexity of matching non-injective general episodes. *Computation and Logic in the Real World*, pages 288–296, 2007.

33   Adam Polak. Why is it hard to beat $O(n^2)$ for longest common weakly increasing subsequence? *Inf. Process. Lett.*, 132:1–5, 2018.

34   Shinichiro Tago, Tatsuya Asai, Takashi Katoh, Hiroaki Morikawa, and Hiroya Inakoshi. EVIS: A fast and scalable episode matching engine for massively parallel data streams. In *Proc. 17th DASFAA*, pages 213–223, 2012.

35   D. E. Willard. Log-logarithmic worst-case range queries are possible in space $\theta(N)$. *Inf. Process. Lett.*, 17(2):81–84, 1983.

36   Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

37   Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In *Proc.31st SODA*, pages 12–29, 2020.