Truly Subquadratic-Time Extension Queries and Periodicity Detection in Strings with Uncertainties

Costas S. Iliopoulos[1] and Jakub Radoszewski*[2]

1 Department of Informatics, King's College London, London, UK
csi@kcl.ac.uk

2 Department of Informatics, King's College London, London, UK; and Institute of Informatics, University of Warsaw, Warsaw, Poland
jrad@mimuw.edu.pl

Abstract

Strings with don't care symbols, also called partial words, and more general indeterminate strings are a natural representation of strings containing uncertain symbols. A considerable effort has been made to obtain efficient algorithms for pattern matching and periodicity detection in such strings. Among those, a number of algorithms have been proposed that behave well on random data, but still their worst-case running time is $\Theta(n^2)$. We present the first truly subquadratic-time solutions for a number of such problems on partial words. We show that n longest common compatible prefix queries (which correspond to longest common extension queries in regular strings) can be answered on-line in $\mathcal{O}(n\sqrt{n \log n})$ time after $\mathcal{O}(n\sqrt{n \log n})$-time preprocessing. We also present $\mathcal{O}(n\sqrt{n \log n})$-time algorithms for computing the prefix array and two types of border array of a partial word. We show how our solutions can be adapted to indeterminate strings over a constant-sized alphabet and prove that, unless the Strong Exponential Time Hypothesis is false, the considered problems cannot be solved efficiently over a general alphabet.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases string with don't cares, partial word, indeterminate string, longest common conservative prefix queries, prefix array

Digital Object Identifier 10.4230/LIPIcs.CPM.2016.8

1 Introduction

In this work we deal with different representations of sequential data with uncertainty and imprecision. An (ideal) text is a sequence of symbols from an alphabet Σ. The symbols at some positions may be unknown; in this case they are represented by a *don't care symbol* (sometimes called a *hole* and denoted as $\diamond$) and the resulting sequence is called a *partial word*. In a more general variant, for some positions, instead of a single character from Σ or a hole, a subset of Σ is specified, thus representing a symbol which can be decoded in a number of ways. The presence of such generalised symbols results in a so-called *indeterminate string* (also called a *degenerate string*).

Our main goal here is to develop worst-case efficient algorithms for different variants of pattern matching problem and periodicities detection in the context of strings with uncertainty. The classical pattern matching problem consists in finding all fragments of a given text that match a given pattern. In the presence of uncertainty one needs to specify

* The author is a Newton International Fellow. Supported by the Polish Ministry of Science and Higher Education under the "Iuventus Plus" program in 2015–2016 grant no 0392/IP3/2015/73.

the relation of *matching* (denoted by $\approx$): a don't care symbol matches every other symbol, and a generalised symbol matches every symbol that belongs to the set represented by it (in particular, two generalised symbols match if their sets have a common element). The pattern matching problem is well-studied in the case of partial words [14, 21, 22, 9, 8]. Also if the pattern is an indeterminate string and the text is a regular string, then worst-case efficient [2] or practically efficient [26, 17, 24] algorithms are known.

One of the variants of the pattern matching problem in strings with uncertainty are *longest common compatible prefix queries* (*lccp-queries*), being a natural generalisation of longest common prefix queries in a regular string. Here we are to preprocess a text of length n with uncertain symbols so that the queries for longest matching prefix of any two suffixes of the text can be answered efficiently. They were first defined in [6], where a solution for partial words was presented with $\mathcal{O}(n^2)$ preprocessing time and $\mathcal{O}(1)$ query time for the case of a constant-sized alphabet. A solution with the same complexity for a linearly-sortable alphabet, which works more efficiently in the case that the number of blocks of don't cares in the text is bounded, was shown in [11]. A connected notion is that of a *prefix array*, which stores the answers to the longest common compatible prefix queries between the whole text and all its suffixes. Its $\mathcal{O}(n^2)$ worst-case time (and $\mathcal{O}(n)$ average time) computation for partial words was shown in [18] and for indeterminate strings in [23]. Further combinatorial insights on the prefix array of an indeterminate string have been recently presented in [3, 7].

The basic array of periodicity on strings is the *border array*. It stores, for every prefix of a string, the length of its longest proper border. Its importance stems from applications in pattern matching algorithms and connections with the set of periods of a string; see [10, 13]. There are two different definitions of border on strings with uncertainty; see [16, 23]. A *quantum* border of an uncertain string X is its prefix that matches its suffix. The main weakness of this definition is that if X has a quantum border of length b, there does not necessarily need to exist a solid string S matching X and having a border of length b. For example, this is the case for $X = \mathtt{a} \diamond \mathtt{b}$ which has a quantum border of length 2: $\mathtt{a}\diamond \approx \diamond\mathtt{b}$; however, none of the strings aab, abb has a border of this length. Therefore, one could be interested in so-called *deterministic* borders: a deterministic border of an uncertain text is defined as a border of some regular string that matches this text. As in the case of regular strings, quantum and deterministic borders correspond to quantum and deterministic *periods* of uncertain texts (the definitions are deferred until Section 2) and thus allow periodicity detection. Quantum periods are also called weak periods and deterministic periods are also called strong periods [5]. Both variants of the border array for a partial word or an indeterminate string can be computed in $\mathcal{O}(n^2)$ worst-case time and $\mathcal{O}(n)$ average time; see [18, 16].

Our Results. In Section 3 we show that, for a partial word of length n, for any $q \in \{1, \dots, n\}$ one can compute in $\mathcal{O}(n^2 \log n/q)$ time a data structure for answering lccp-queries in $\mathcal{O}(q)$ time. In particular, one can answer n such queries in a partial word in $\mathcal{O}(n\sqrt{n \log n})$ time. In Section 4 we present a construction of the prefix array and both types of a border array – hence, the corresponding types of period array – in the same time complexity. Finally in Section 5 we show that all these results (apart from the deterministic border/period array computation) extend to indeterminate strings over a constant-sized alphabet. Under the word-RAM model the complexities improve by a factor of $\sqrt{\log n}$. We also argue that, under the Strong Exponential Time Hypothesis, none of the considered problems can be solved on indeterminate strings in $\mathcal{O}(n^{2-\epsilon}\sigma^{\mathcal{O}(1)})$ time over an alphabet of size σ, for $\epsilon > 0$.

2 Preliminaries

A *string* S of length $n = |S|$ is a sequence of n letters over a finite alphabet Σ. The letter at the position i, for $1 \le i \le n$, is denoted as $S[i]$. The size of the alphabet is denoted by $\sigma = |\Sigma|$. By $S[i..j]$ we denote a *factor* of S equal to $S[i] \dots S[j]$ (if $i > j$ then it is the empty string ε). The factor is called a *prefix* if $i = 1$ and a *suffix* if $j = n$. The length of the longest common prefix of $S[i..n]$ and $S[j..n]$ is denoted as $\mathsf{lcp}(i, j)$.

If $S[1..b] = S[n-b+1..n]$ then the string $S[1..b]$ is called a *border* of S. A positive integer $p \le n$ is called a *period* of S if $S[i] = S[i+p]$ for all $i = 1, \dots, n-p$. It is known that S has a period p if and only if it has a border of length $n - p$ [10, 13].

For a string S we define the following arrays of length n:

- prefix array π, such that $\pi[i] = \mathsf{lcp}(1, i)$ for $i \ge 2$;
- border array B, such that $B[i]$ is the length of the longest border of $S[1..i]$;
- period array P, such that $P[i]$ is the shortest period of $S[1..i]$.

A *partial word* X of length $n = |X|$ is a sequence of elements $X[1], \dots, X[n]$ from $\Sigma \cup \{\diamond\}$. Here $\diamond \notin \Sigma$ is a special character called a *don't care symbol.* Two characters $a, b \in \Sigma \cup \{\diamond\}$ are said to match (denoted as $a \approx b$) if $a = b$ or $a = \diamond$, or $b = \diamond$. The $\approx$-relation is extended to partial words position by position. Note that $\approx$ is not transitive; for instance, $\mathtt{a} \approx \diamond$ and $\diamond \approx \mathtt{b}$, but $\mathtt{a} \not\approx \mathtt{b}$.

We define a *factor* of X as a partial word $X[i..j] = X[i] \dots X[j]$ (if $i > j$ then it is the empty partial word). A factor is called a *prefix* if $i = 1$ and a *suffix* if $j = n$. The length of the longest common conservative prefix at positions i and j, denoted as $\mathsf{lccp}(i, j)$, is the greatest integer k such that $X[i..i+k-1] \approx X[j..j+k-1]$. Then the prefix array $\pi[2..n]$ of X is defined as $\pi[i] = \mathsf{lccp}(1, i)$.

A *quantum border* of a partial word X is an integer $b \in \{0, \dots, n\}$ such that $X[1..b] \approx X[n-b+1..n]$. A *quantum period* of X is an integer $p \in \{0, \dots, n\}$ such that $X[i] \approx X[i+p]$ for all $i = 1, \dots, n-p$. Those two notions correspond, i.e., if X has quantum period p then it has a quantum border $n - p$ and *vice versa*; see [23]. A *deterministic border* (*deterministic period*) of X is an integer b (p, respectively) such that there exists a string S such that $S \approx X$ and S has a border of length b (a period p, respectively). Here, obviously, we have that if p is a deterministic period of X, then $n - p$ is a deterministic border of X and *vice versa*. Up to the length $\frac{n}{2}$ quantum and deterministic borders of a partial word are the same [16]. However, as we have mentioned before, this does not apply to greater lengths. For partial words we have the following alternative definition of a deterministic period.

▶ **Observation 1.** *A positive integer p is a deterministic period of a partial word X if and only if $X[i] \approx X[j]$ whenever $p \mid i - j$.*

▶ **Example 2.** The partial word

$$\mathtt{a\,b\,a} \diamond \diamond \diamond \mathtt{a} \diamond \mathtt{a\,a}$$

has six quantum periods: 2, 3, 4, 6, 9, 10. For example, 2 is a quantum period because

$$\mathtt{a\,b} \approx \mathtt{a}\diamond \approx \diamond\diamond \approx \mathtt{a}\diamond \approx \mathtt{a\,a}.$$

However, this partial word has only four deterministic periods 3, 6, 9, 10, all corresponding to the solid string

$$\mathtt{aba\,aba\,aba\,a}.$$

As in the case of regular strings, we introduce the border arrays and the period arrays for partial words. By $QB[i]$, $QP[i]$, $DB[i]$, and $DP[i]$ we denote the longest quantum border, shortest quantum period, longest deterministic border, and shortest deterministic period of $X[1..i]$. As we have already mentioned, for every i it holds that $QP[i] = i - QB[i]$ and $DP[i] = i - DB[i]$.

▶ **Example 3.** The following table presents the prefix array and the border arrays of two types of an example partial word.

$X[i]$	a	◇	a	◇	b	a	b	b	b	◇
$\pi[i]$	–	4	2	5	0	2	0	0	0	1
$QB[i]$	0	1	2	3	4	3	4	5	0	1
$DB[i]$	0	1	2	3	2	3	2	0	0	1

We say that a pattern P *occurs* in a text T, both being partial words, at position i if $P \approx T[i..i+|P|-1]$. Pattern matching on partial words can be done efficiently via convolutions. A line of research lead through alphabet-dependent algorithms and randomized algorithms [14, 21, 22] eventually to an efficient deterministic algorithm; see [9, 8].

▶ **Fact 4.** *Given two partial words P and T of length m and n, respectively, one can find all occurrences of P in T in $\mathcal{O}(n \log m)$ time.*

3 Longest Common Compatible Prefix Queries

Let X be a partial word of length n. In this section we show how to answer lccp-queries for X in $\mathcal{O}(q)$ time after $\mathcal{O}(n^2 \log n/q)$-time preprocessing, for any $q \in \{1, \ldots, n\}$. In the solution we use a dynamic programming approach combined with pattern matching in partial words.

Let us define a family of partial words $X_i = X[(i-1)q+1..iq]$ for $i = 1, \ldots, \lfloor n/q \rfloor$. Let the array $A[i,j]$ for $i = 1, \ldots, \lfloor n/q \rfloor$ and $j = 1, \ldots, n-q+1$ be defined as follows: $A[i,j] = 1$ if $X_i \approx X[j..j+q-1]$, and $A[i,j] = 0$ otherwise.

▶ **Observation 5.** *The array A can be computed in $\mathcal{O}(\frac{n^2 \log n}{q})$ time.*

Proof. Computation of the array is equivalent to pattern matching of each X_i in X. The time complexity follows from Fact 4. ◀

Let the array L for $i = 1, \ldots, \lfloor n/q \rfloor$ and $j = 1, \ldots, n-q+1$ be defined as follows:

$$L[i,j] = \max\{k \geq 0 \,:\, X_i \ldots X_{i+k-1} \approx X[j..j+kq-1]\}.$$

▶ **Lemma 6.** *The array L can be computed from the array A in $\mathcal{O}(\frac{n^2}{q})$ time.*

Proof. We compute $L[i,j]$ for decreasing values of i and j using a dynamic programming approach. Assume that if $i > \lfloor n/q \rfloor$ or $j > n-q+1$, then $L[i,j] = 0$. For $i = \lfloor n/q \rfloor, \ldots, 1$ and $j = n-q+1, \ldots, 1$, if $A[i,j] = 1$, then $L[i,j] = L[i+1, j+q] + 1$, and otherwise $L[i,j] = 0$. ◀

We answer lccp-queries using the array L. In the query algorithm we use a simple *bounded* lccp routine (denoted as blccp) that for a pair of indices i, j and a length parameter ℓ returns $\min(\mathsf{lccp}(i,j), \ell)$.

▶ **Observation 7.** *blccp(i,j,ℓ) for any i,j,ℓ can be computed in $\mathcal{O}(\ell)$ time.*

```
function lccp($i, j$)
$k := \lfloor i/q \rfloor$;  $\ell := kq - i + 1$;
$x :=$ blccp($i, j, \ell$);
if $x < \ell$ then return $x$;
$y := L[k+1, j+x] \cdot q$;
$z :=$ blccp($i+x+y, j+x+y, q$);
return $x+y+z$;
```

Figure 1 Function $\mathsf{lccp}(i, j)$.

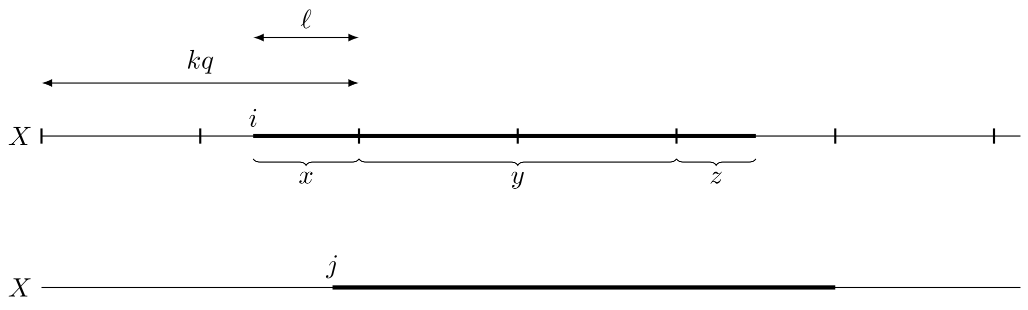

Figure 2 A schematic illustration of the algorithm answering an $\mathsf{lccp}(i, j)$-query. For simplicity the partial word X is depicted twice; the upper copy is divided into fragments of length q. The result of the query is shown in bold.

▶ **Lemma 8.** *Knowing the array L for the partial word X, one can compute $\mathsf{lccp}(i, j)$ for any $i, j \in \{1, \ldots, n\}$ in $\mathcal{O}(q)$ time.*

Proof. The $\mathsf{lccp}(i, j)$ query is answered by the algorithm from the pseudocode in Figure 1. First, we find the smallest ℓ such that $i + \ell \equiv 1 \pmod{q}$. We start with an lccp-query from i and j bounded by ℓ (part x). If the bound is attained, we read the remaining lccp length up to a multiple of q from the array L (part y). The remainder of the result modulo q is computed using a final blccp query (part z); see also Figure 2.

The only non-constant-time operations are two blccp-queries, which can be answered in $\mathcal{O}(q)$ time each by Observation 7. ◀

▶ **Theorem 9.** *Let X be a partial word of length n and $q \in \{1, \ldots, n\}$ be an integer. After $\mathcal{O}(n^2 \log n/q)$-time and $\mathcal{O}(n^2/q)$-space preprocessing one can answer lccp-queries for X in $\mathcal{O}(q)$ time.*

Proof. We use Observation 5 and Lemma 6 for the construction of the data structure and the algorithm of Lemma 8 for answering lccp-queries. ◀

4 Computing Periodicity Arrays

The prefix array of a partial word can be computed via n lccp-queries. By selecting $q = \lfloor \sqrt{n \log n} \rfloor$ in Theorem 9, we obtain $\mathcal{O}(n\sqrt{n \log n})$-time computation of the array. The space usage of this algorithm is $\mathcal{O}(n\sqrt{n/\log n})$. However, we can obtain better space complexity if we refrain from storing the whole array L.

▶ **Corollary 10.** *The prefix array of a partial word of length n can be computed in $\mathcal{O}(n\sqrt{n\log n})$ time and $\mathcal{O}(n)$ space.*

Proof. Consider the array L from the algorithm of Theorem 9. To compute the array π, it suffices to store the values $\ell_j = L[1, j]$ for $j = 2, \ldots, n$ (assuming $L[1, j] = 0$ for $j > n-q+1$). Then

$$\pi[j] \;=\; \ell_j \cdot q \;+\; \mathsf{blccp}(1 + \ell_j \cdot q,\, j + \ell_j \cdot q,\, q),$$

which can be computed in $\mathcal{O}(q)$ time.

The values ℓ_j can be computed with only linear space. Probably the simplest approach is to perform subsequent matching in X of $\lfloor n/q \rfloor$ partial word patterns of the form $X_1 \ldots X_i$ for $i = 1, \ldots, \lfloor n/q \rfloor$. Then as ℓ_j we store the greatest index i such that $X_1 \ldots X_i$ occurs at the position j in X.

By Fact 4, the aforementioned computation of ℓ_j-values takes $\mathcal{O}(n^2 \log n/q)$ time. Knowing those values, we can compute all $\pi[j]$ in $\mathcal{O}(nq)$ time. We select $q = \sqrt{n\log n}$ and obtain an $\mathcal{O}(n\sqrt{n\log n})$-time algorithm. It requires only linear space. ◀

In the case of solid strings one can compute the border array from the prefix array in linear time; see [10, 13]. For partial words we can apply a similar approach to compute the quantum border array. Assume $\pi[n+1] = 0$. We use the following combinatorial observation.

▶ **Observation 11.** *p is a quantum period of $X[1..i]$ if and only if $p \le i \le p + \pi[p+1]$.*

Proof.
(⇒) Assume that p is a quantum period of $X[1..i]$. Then $i-p$ is a quantum border of $X[1..i]$, $X[1..i-p] \approx X[p+1..i]$. Hence, $\pi[p+1] \ge i-p$, i.e., $i \le p + \pi[p+1]$. Obviously, $p \le i$.

(⇐) We have $X[1..\pi[p+1]] \approx X[p+1..p+1+\pi[p+1]-1]$. As $p \le i \le p + \pi[p+1] = p+1+\pi[p+1]-1$, this concludes that $X[1..i-p] \approx X[p+1..i]$. Hence, $i-p$ is a quantum border of $X[1..i]$, so p is a quantum period of $X[1..i]$. ◀

▶ **Lemma 12.** *The quantum border array and the quantum period array of a partial word of length n can be computed in $\mathcal{O}(n)$ time given its prefix array.*

Proof. We focus on computing the array $QP[i]$; the array $QB[i]$ can then be computed in $\mathcal{O}(n)$ time. The algorithm is shown in Figure 3.

In the algorithm we store the last index l for which $QP[l]$ has been computed. For every $p \in \{1, \ldots, n\}$ we set the value of the quantum period to p for positions determined by Observation 11, taking care not to override the previously computed values. As each position in QP is set at most once, the algorithm runs in linear time. ◀

Let us proceed to the computation of deterministic border and period arrays. We will use the following characterisation of a deterministic period of a partial word in terms of its quantum periods, which is a consequence of Observation 1.

▶ **Observation 13.** *A partial word X has a deterministic period p if and only if it has all quantum periods jp for $1 \le j \le \frac{n}{p}$.*

Let us define

$$I_k(p) = [kp, (k+1)p), \quad M_k(p) = \min_{j=1,\ldots,k} (jp + \pi[jp+1]).$$

We combine Observation 11 with Observation 13 to obtain the following criterion.

```
function Compute-$QP(X, n)$
{ Assume $\pi[n+1] = 0$ }
$l := 0$;
for $p := 1$ to $n$ do
    for $i := \max(p, l+1)$ to $p + \pi[p+1]$ do
        $QP[i] := p$;
    $l := \max(l,\ p + \pi[p+1])$;
return $QP$;
```

Figure 3 Algorithm computing the quantum period array.

▶ **Observation 14.** *If $i \in I_k(p)$, then $X[1..i]$ has a deterministic period p if and only if $i \le M_k(p)$.*

Using Observation 14 we obtain the following result.

▶ **Lemma 15.** *The deterministic border array and the deterministic period array of a partial word X can be computed in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space given its prefix array.*

Proof. First we compute, for every $p \in \{1, \dots, n\}$, an interval $I(p)$ such that $i \in I(p)$ if and only if $X[1..i]$ has a deterministic period p. For this, notice that the intervals $I_k(p)$ for $k = 1, \dots, \left\lfloor \frac{n}{p} \right\rfloor$ are pairwise disjoint, their left endpoints are monotonically increasing, whereas the values $M_k(p)$ for $k = 1, \dots, \left\lfloor \frac{n}{p} \right\rfloor$ are monotonically non-increasing. By Observation 14, we have $I_k(p) \subseteq I(p)$ as long as $M_k(p) \ge (k+1)p - 1$. The last interval included in $I(p)$ is $I_k(p) \cap [1, M_k(p)]$ for the smallest k such that $M_k(p) < (k+1)p - 1$, if such a value of k exists. The computation of $I(p)$ takes $\mathcal{O}(\frac{n}{p})$ time, which gives $\mathcal{O}\left(\sum_{p=1}^{n} \frac{n}{p}\right) = \mathcal{O}(n \log n)$ time in total.

The final step consists in computing the smallest deterministic period of each $X[1..i]$. This is equivalent to the min-variant of the Manhattan skyline problem: for a family of intervals $I(p)$ with heights p we are to compute, for every i, the smallest height of an interval that covers it. Using the linear-time nested union/find data structure [15] this problem can be solved in $\mathcal{O}(n)$ time (see also Section 5.1 in [12]). ◀

We plug Corollary 10 into Lemmas 12 and 15 to arrive at the following final result.

▶ **Theorem 16.** *The prefix array, the quantum border array, the quantum period array, the deterministic border array, and the deterministic period array of a partial word of length n can all be computed in $\mathcal{O}(n\sqrt{n \log n})$ time and $\mathcal{O}(n)$ space.*

▶ Remark. In [18] it is mentioned that all quantum periods/borders of the whole partial word can be computed via a single run of pattern matching, i.e., in $\mathcal{O}(n \log n)$ time. Therefore, by Observation 13, all deterministic periods (hence, borders) of the whole partial word can also be computed in $\mathcal{O}\left(\sum_{p=1}^{n} \frac{n}{p}\right) = \mathcal{O}(n \log n)$ time (and linear space).

5 The Case of Constant Alphabet and Indeterminate Strings

An *indeterminate string* X of length $|X| = n$ over an alphabet Σ of size σ is a sequence of nonempty sets $X[1], \dots, X[n]$ with $X[i] \subseteq \Sigma$. Two subsets A, B of Σ are said to match (denoted as $A \approx B$) if they contain at least one letter in common. Under this matching relation one can transfer all notions of pattern matching and periodicity from partial words to indeterminate strings [16, 23]. In this section we show that the majority of the results from the previous sections extend to indeterminate strings over a constant-sized alphabet. Due to large constants hidden in the time complexities, the resulting algorithms are plausible in practice only for a small σ. The most common alphabet over which indeterminate strings are considered is $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Such indeterminate strings occur, e.g., in the FASTA format.

In the data structure of Section 3 we used an efficient pattern matching routine on partial words. The state-of-the-art algorithm for pattern matching on indeterminate strings works in $\mathcal{O}(\sigma n \log n)$ time or in $\mathcal{O}(n\sqrt{n \log n})$ time [2], however, only if the text is a *regular string*. If both the pattern and the text are indeterminate, we obtain an efficient solution for $\sigma = \mathcal{O}(1)$.

▶ **Lemma 17.** *Given two indeterminate strings P and T of length m and n, respectively, over a constant-sized alphabet, one can find all occurrences of P in T in $\mathcal{O}(n \log m)$ time.*

Proof. For every $A \subseteq \Sigma$ we perform the following procedure. Construct a binary string P' of length m such that $P'[i] = 1$ if and only if $P[i] = A$. Construct a binary string T' of length n such that $T'[i] = 1$ if and only if the sets $T[i]$ and A are disjoint. Use an FFT convolution to count, for every alignment of P' and T', the number of common 1s at the corresponding positions of P' and a factor of T'.

In the end we report all alignments for which no common 1 was found in any of the steps. The algorithm works in 2^σ steps, each taking $\mathcal{O}(n \log m)$ time. ◀

Another building block of the lccp data structure are the blccp queries. For indeterminate strings with $\sigma = \mathcal{O}(1)$ they can be implemented in $\mathcal{O}(\ell)$ time just as in Observation 7. We can also answer them slightly faster using standard properties of the word-RAM model.

▶ **Fact 18.** *For an indeterminate string X of length n over an alphabet of size $\sigma = \mathcal{O}(1)$, after $\mathcal{O}(n)$-time and space preprocessing one can compute $\mathsf{blccp}(i, j, \ell)$ in $\mathcal{O}(\ell / \log n)$ time.*

Proof. Consider any $\epsilon > 0$. Let $c = (2+\epsilon)\sigma$ and $L = \max\left(\left\lfloor \frac{\log n}{c} \right\rfloor, 1\right)$. The number of indeterminate strings of length L over the alphabet of size σ is:

$$2^{\sigma L} \le 2^{\frac{\sigma \log n}{(2+\epsilon)\sigma}} = n^{\frac{1}{2+\epsilon}} < \sqrt{n},$$

so each of them can be assigned an integer identifier between 1 and $\lfloor \sqrt{n} \rfloor$. For every pair of indeterminate strings of length L we precompute their lccp. There are $2^{2L\sigma}$ such pairs, and the result for each of them can be computed in $\mathcal{O}(L)$ time. All the results can be stored in an array of size $2^{2L\sigma}$. In total this precomputation takes

$$\mathcal{O}(2^{2L\sigma} L) = \mathcal{O}(n^{\frac{1}{1+\epsilon/2}} \log n) = o(n)$$

time.

For every factor of X of length L we then compute its integer identifier. This can be done in $\mathcal{O}(n)$ time if the identifiers are determined by Rabin-Karp-style polynomials with the rolling property; see [13]. Finally a $\mathsf{blccp}(i, j, \ell)$ query is answered by cutting the factors of length ℓ into factors of length L and using the precomputed answers. ◀

▶ Remark. For a partial word over a constant-sized alphabet a much better constant $c = (2+\epsilon)\log(\sigma+1)$ would suffice.

Using Lemma 17 and Fact 18 we obtain an implementation of lccp-queries on indeterminate strings.

▶ **Theorem 19.** *Let X be an indeterminate string of length n over a constant-sized alphabet and $q \in \{1,\ldots,\lfloor n/\log n \rfloor\}$ be an integer. After $\mathcal{O}(n^2/q)$-time preprocessing one can answer* lccp*-queries for X in $\mathcal{O}(q)$ time.*

Now the computation of the prefix array and quantum border/period array is the same as in partial words. However, the computation of the deterministic border and period array does not generalise, since Observation 1, and consequently Observation 13, does not hold for indeterminate strings. For example, consider an indeterminate string X of length 3 over $\Sigma = \{\mathtt{a},\mathtt{b},\mathtt{c}\}$ such that $X[1] = \{\mathtt{a},\mathtt{b}\}$, $X[2] = \{\mathtt{a},\mathtt{c}\}$, $X[3] = \{\mathtt{b},\mathtt{c}\}$. It has a quantum period 1 and $X[1] \approx X[2] \approx X[3] \approx X[1]$. However, it does not have a deterministic period 1 since there is no $s \in \Sigma$ that would match $X[1]$, $X[2]$, and $X[3]$ simultaneously. Therefore we obtain only the following result for indeterminate strings.

▶ **Corollary 20.** *The prefix array, the quantum border array, and the quantum period array of an indeterminate string of length n over a constant-sized alphabet can be computed in $\mathcal{O}(n\sqrt{n})$ time and $\mathcal{O}(n)$ space.*

The time complexities of the algorithms of Corollary 20 have exponential dependency on the alphabet size σ. We will now show that, under some well-known hypotheses, no truly subquadratic algorithms with polynomial dependency on σ exist for any of the considered problems.

The *Orthogonal Vectors Problem* is defined as follows: given two sets A and B containing N vectors from $\{0,1\}^d$ each, does there exist a pair of vectors $\alpha \in A$ and $\beta \in B$ that is orthogonal, i.e., $\sum_{h=1}^{d} \alpha[h]\beta[h] = 0$? The following conjecture is known to be implied (see [25]) by the Strong Exponential Time Hypothesis (SETH), see [19, 20], which asserts that for any $\epsilon > 0$ there is an integer $k > 3$ such that k-SAT cannot be solved in $2^{(1-\epsilon)n}$ time. This conjecture has already been applied to prove hardness results of stringology problems [1, 4].

▶ **Conjecture 21.** *There is no $\epsilon > 0$ and an algorithm that solves the Orthogonal Vectors Problem in $\mathcal{O}(N^{2-\epsilon} \cdot d^{\mathcal{O}(1)})$ time.*

We will show that, under this conjecture, pattern matching on indeterminate strings of length n and $2n$, respectively, both over an alphabet of size σ, cannot be solved in $\mathcal{O}(n^{2-\epsilon} \cdot \sigma^{\mathcal{O}(1)})$ time.

▶ **Theorem 22.** *The Orthogonal Vectors Problem can be reduced to pattern matching of an indeterminate pattern of length n in an indeterminate text of length $2n$, where $n = N$, over an alphabet of size $\sigma = d$.*

Proof. Let $A = \{\alpha_1,\ldots,\alpha_N\}$ and $B = \{\beta_1,\ldots,\beta_N\}$ be the two sets of vectors in $\{0,1\}^d$. Consider an alphabet $\Sigma = \{1,\ldots,\sigma\}$. For a vector $\alpha \in \{0,1\}^d$, by $f(\alpha)$ we denote the subset of Σ defined as: $s \in f(\alpha) \Leftrightarrow \alpha[s] = 1$. Under this mapping, two vectors α and β are orthogonal if and only if the sets $f(\alpha)$ and $f(\beta)$ are disjoint, i.e., the indeterminate symbols $f(\alpha)$ and $f(\beta)$ *do not* match.

We construct an indeterminate pattern $P = f(\alpha_1)\ldots f(\alpha_N)$ and an indeterminate text $T = f(\beta_1)\ldots f(\beta_N)f(\beta_1)\ldots f(\beta_N)$. Then the Orthogonal Vectors Problem for the sets A and B has a positive answer if and only if P *does not* occur in T at any of the positions $1,\ldots,n$. ◀

Let P and T be the indeterminate pattern and text of Theorem 22 and S be the concatenation of P and T. As the pattern matching can be solved by computing the prefix array of S or any of the border arrays of S, or answering n lccp-queries in S, we obtain the following conditional lower bound.

▶ **Corollary 23.** *The prefix array and any of the border arrays of an indeterminate string of length n cannot be computed in $\mathcal{O}(n^{2-\epsilon} \cdot \sigma^{\mathcal{O}(1)})$ time unless SETH fails. Also the problem of answering n lccp-queries in an indeterminate string of length n cannot be solved in $\mathcal{O}(n^{2-\epsilon} \cdot \sigma^{\mathcal{O}(1)})$ time unless SETH fails.*

6 Conclusions and Final Remarks

We have presented a worst-case efficient framework for answering longest common compatible prefix queries in a partial word. We have then shown how we can compute the prefix array and two types of border/period arrays of a partial word basically as fast as answering n lccp-queries. In some cases lccp-queries can be answered faster than using our approach – e.g., if the number of don't care symbols is small or the number of groups of consecutive don't care symbols is small, see [11] – which automatically yields more efficient algorithms for computing the aforementioned arrays.

Then we have presented extensions of all the results apart from the construction of the deterministic border and period array to indeterminate strings over a constant-sized alphabet. We have also argued that, for general alphabets, efficient solutions to any of the considered problems for indeterminate strings would violate the Strong Exponential Time Hypothesis. This, in particular, justifies the usage of heuristic approaches for these problems. As an open question we leave the computation of deterministic periods of an indeterminate string over a constant-sized alphabet in $\mathcal{O}(n^{2-\epsilon})$ time.

Acknowledgements. The authors thank an anonymous referee for a number of helpful suggestions.

References

1 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. `doi:10.1109/FOCS.2015.14`.

2 Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987. `doi:10.1137/0216067`.

3 Ali Alatabbi, M. Sohel Rahman, and William F. Smyth. Inferring an indeterminate string from a prefix graph. *J. Discrete Algorithms*, 32:6–13, 2015. `doi:10.1016/j.jda.2014.12.006`.

4 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58. ACM, 2015. `doi:10.1145/2746539.2746612`.

5 Francine Blanchet-Sadri and Robert A. Hegstrom. Partial words and a theorem of Fine and Wilf revisited. *Theor. Comput. Sci.*, 270(1-2):401–419, 2002. `doi:10.1016/S0304-3975(00)00407-2`.

6 Francine Blanchet-Sadri and Justin Lazarow. Suffix trees for partial words and the longest common compatible prefix problem. In Adrian Horia Dediu, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications*

– *7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings*, volume 7810 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2013. doi:10.1007/978-3-642-37064-9_16.

7 Manolis Christodoulakis, P. J. Ryan, William F. Smyth, and Shu Wang. Indeterminate strings, prefix arrays & undirected graphs. *Theor. Comput. Sci.*, 600:34–48, 2015. doi:10.1016/j.tcs.2015.06.056.

8 Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007. doi:10.1016/j.ipl.2006.08.002.

9 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 592–601. ACM, 2002. doi:10.1145/509907.509992.

10 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, 2007.

11 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Alessio Langiu, Jakub Radoszewski, Wojciech Rytter, Bartosz Szreder, and Tomasz Waleń. A note on the longest common compatible prefix problem for partial words. *J. Discrete Algorithms*, 34:49–53, 2015. doi:10.1016/j.jda.2015.05.003.

12 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Extracting powers and periods in a word from its runs structure. *Theor. Comput. Sci.*, 521:29–41, 2014. doi:10.1016/j.tcs.2013.11.018.

13 Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology*. World Scientific, 2003.

14 Michael J. Fischer and Michael S. Paterson. String matching and other products. In Richard Karp, editor, *Proceedings of the 7th SIAM-AMS Complexity of Computation*, pages 113–125, 1974. doi:10.1007/978-3-540-89097-3_14.

15 Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comput. Syst. Sci.*, 30(2):209–221, 1985. doi:10.1016/0022-0000(85)90014-5.

16 Jan Holub and William F. Smyth. Algorithms on indeterminate strings. In *Proceedings of 14th Australasian Workshop on Combinatorial Algorithms*, pages 36–45, 2003.

17 Jan Holub, William F. Smyth, and Shu Wang. Fast pattern-matching on indeterminate strings. *J. Discrete Algorithms*, 6(1):37–50, 2008. doi:10.1016/j.jda.2006.10.003.

18 Costas S. Iliopoulos, Manal Mohamed, Laurent Mouchard, Katerina Perdikuri, William F. Smyth, and Athanasios K. Tsakalidis. String regularities with don't cares. *Nord. J. Comput.*, 10(1):40–51, 2003.

19 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

20 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

21 Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *39th Annual Symposium on Foundations of Computer Science, FOCS'98, November 8-11, 1998, Palo Alto, California, USA*, pages 166–173. IEEE Computer Society, 1998. doi:10.1109/SFCS.1998.743440.

22 Adam Kalai. Efficient pattern-matching with don't cares. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 655–656. ACM/SIAM, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545468.

23 William F. Smyth and Shu Wang. New perspectives on the prefix array. In Amihood Amir, Andrew Turpin, and Alistair Moffat, editors, *String Processing and Information Retrieval,* *15th International Symposium, SPIRE 2008, Melbourne, Australia, November 10-12, 2008. Proceedings*, volume 5280 of *Lecture Notes in Computer Science*, pages 133–143. Springer, 2008. doi:10.1007/978-3-540-89097-3_14.

24 William F. Smyth and Shu Wang. An adaptive hybrid pattern-matching algorithm on indeterminate strings. *Int. J. Found. Comput. Sci.*, 20(6):985–1004, 2009. doi:10.1142/S0129054109007005.

25 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

26 Sun Wu and Udi Manber. Agrep – a fast approximate pattern-matching tool. In *Proceedings USENIX Winter 1992 Technical Conference*, page 153–162, 1992. URL: https://www.usenix.org/legacy/publications/library/proceedings/wu.pdf.