Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Covering problems for partial words and for indeterminate strings

Maxime Crochemore [a,b], Costas S. Iliopoulos [a], Tomasz Kociumaka [c], Jakub Radoszewski [a,c,*], Wojciech Rytter [c], Tomasz Waleń [c]

[a] *Department of Informatics, King's College London, UK*
[b] *Université Paris-Est, France*
[c] *Institute of Informatics, University of Warsaw, Poland*

ARTICLE INFO

Article history:
Received 24 February 2017
Received in revised form 29 April 2017
Accepted 7 May 2017
Available online xxxx

Keywords:
Cover of a word
Partial word
String with don't cares
Indeterminate string
Fixed-parameter tractability

ABSTRACT

Indeterminate strings are a subclass of non-standard words having non-deterministic nature. In a classic string every position contains exactly one symbol—we say it is a *solid* symbol—while in an indeterminate string a position may contain a set of symbols (possible at this position); such sets are called *non-solid* symbols. The most important subclass of indeterminate strings are partial words, where each non-solid symbol is the whole alphabet; in this case non-solid symbols are also called *don't care* symbols. We consider the problem of finding a *shortest cover* of an indeterminate string, i.e., finding a shortest *solid* string whose occurrences cover the whole indeterminate string. We show that this classical problem becomes NP-complete for indeterminate strings and even for partial words. The proof of this fact is one of the main results of this paper. Our other main results focus on design of algorithms efficient with respect to certain parameters of the input (so called FPT algorithms) for the shortest cover problem. For the indeterminate string covering problem we obtain an $O(nk^2 + 2^k k^3)$-time algorithm, where k is the number of non-solid symbols, while for the partial word covering problem we obtain a running time of $O(nk^2 + 2^{O(\sqrt{k}\log k)})$. Additionally, we prove that, unless the Exponential Time Hypothesis is false, no $2^{o(\sqrt{k})}n^{O(1)}$-time solution exists for either problem, which shows that our algorithm for partial words is close to optimal. We also present an algorithm for both problems parameterized both by k and the alphabet size with a simple implementation.

A preliminary version of this article was presented at the 25th International Symposium on Algorithms and Computation (ISAAC 2014), LNCS, vol. 8889, pp. 220–232, Springer (2014) [12].

1. Introduction

A classic string is a sequence of symbols from a given alphabet Σ. In an *indeterminate* string, some positions may contain, instead of a single symbol from Σ (called a *solid* symbol), a non-empty subset of Σ. Such a *non-solid* symbol can mean that

[*] Corresponding author.
E-mail addresses: maxime.crochemore@kcl.ac.uk (M. Crochemore), c.iliopoulos@kcl.ac.uk (C.S. Iliopoulos), kociumaka@mimuw.edu.pl (T. Kociumaka), jrad@mimuw.edu.pl (J. Radoszewski), rytter@mimuw.edu.pl (W. Rytter), walen@mimuw.edu.pl (T. Waleń).

http://dx.doi.org/10.1016/j.tcs.2017.05.026
0304-3975/© 2017 Elsevier B.V. All rights reserved.

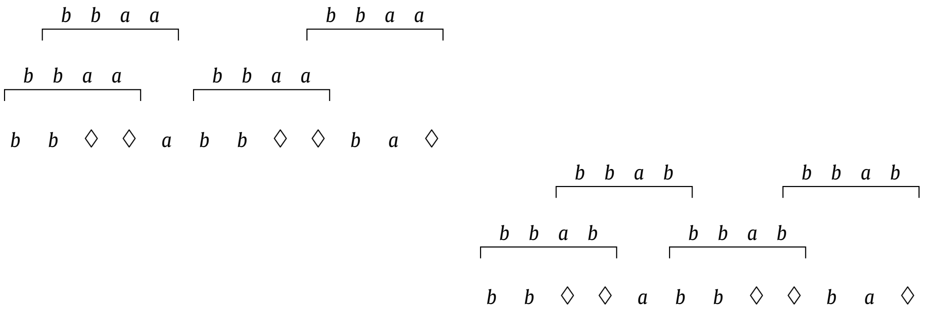


Fig. 1. Partial word $bb\Diamond\Diamond abb\Diamond\Diamond ba\Diamond = bb\{a,b\}\{a,b\}abb\{a,b\}\{a,b\}ba\{a,b\}$ with its two shortest covers.

the exact symbol at the given position is not known, but is suspected to be one of the specified symbols. The simplest type of indeterminate strings are *partial words*, in which every non-solid symbol is a don't care symbol, denoted here $\Diamond$ (other popular notation is $*$), which represents the whole alphabet Σ.

Motivations for indeterminate strings can be found in computational biology, musicology, and other areas. In computational biology, analogous juxtapositions may count as matches in protein sequences. In fact, the FASTA format[1] representing nucleotide or peptide sequences specifically includes indeterminate letters. In music, single notes may match chords, or notes separated by an octave may match; see [16].

Algorithmic study of indeterminate strings is mainly devoted to pattern matching. The first efficient algorithm was proposed by Fischer and Paterson for strings with don't care symbols [13]. Faster algorithms for this case were afterwards given in [29,22,23,10,9]. Pattern matching for general indeterminate strings, known as generalized string matching, was first considered by Abrahamson [1] in the variant that the text is solid. In the most general variant, pattern matching on indeterminate strings probably cannot be solved efficiently in general [19]. There were practical approaches to the problem; see [16,30] for some recent examples. A survey on partial words, related mostly to their combinatorics, can be found in a book by Blanchet-Sadri [7].

The notion of *cover* belongs to the area of quasiperiodicity, that is, a generalization of periodicity in which the occurrences of the period may overlap [4]. A cover of a solid string S is a string that covers all positions of S with its occurrences. Covers in solid strings were already extensively studied. A linear-time algorithm finding the shortest cover of a string was given by Apostolico et al. [5] and later on improved into an on-line algorithm by Breslauer [8]. A linear-time algorithm computing all the covers of a string was proposed by Moore and Smyth [28]. Afterwards an on-line algorithm computing all the covers of a string was given by Li and Smyth [26].

Other types of quasiperiodicities are seeds [18,24] and numerous variants of covers and seeds, including approximate and partial covers and seeds.

Classical periodicities, including periods and the border array, were already extensively studied in the field of indeterminate strings (originating from the papers [15,17]) and also for partial words (see, e.g., the book of Blanchet-Sadri [7]). Quasiperiodicity is a relaxation of the classical notions of periodicity and, as such, enables one to find approximate repetitive structures. This is important in imprecise data that occurs often in such areas of applications as computational biology and music.

The problem that we consider in this work is as follows (see also Fig. 1):

Covering an indeterminate string
Input: an indeterminate string T
Output: the length of a shortest solid cover of T

We allow the same non-solid symbol to match two different solid symbols for two different occurrences of the same cover. All our algorithms can actually recover an example shortest cover.

Throughout the paper we use the following notations: n is the length of the given indeterminate string T, k is the number of non-solid symbols in T, and σ is the size of the underlying alphabet Σ. We assume that $2 \le \sigma \le n$ with $\Sigma = \{1, \dots, \sigma\}$ and that each non-solid symbol in the indeterminate string is represented by a bit vector of size σ. Thus the size of the input is $O(n + \sigma k)$.

The first attempts to the problem of indeterminate string covering were made in [3,6,17]. The common assumption of these papers is that $\sigma = O(1)$. The earliest contribution [17] does not provide a rigorous definition of a cover of an

[1] http://en.wikipedia.org/wiki/FASTA_format.

Please cite this article in press as: M. Crochemore et al., Covering problems for partial words and for indeterminate strings, Theoret. Comput. Sci. (2017), http://dx.doi.org/10.1016/j.tcs.2017.05.026

indeterminate string. Using our definition, their algorithm runs in $O(n^2 \log n)$ worst-case time and works only in the case that every border of the text is solid. In [3,6] a different definition of a cover than ours is used, allowing for the presence of indeterminate symbols in a cover. The contribution [3] considers only so-called conservative indeterminate strings, for which $k = O(1)$, and addresses a problem of checking if its border of a given length ℓ is also its cover. They construct an automaton for all solid strings that match the border, being potentially of size $O(\ell\sigma^k)$, and then use it to perform pattern matching on the indeterminate text in time $O(n\ell\sigma^k)$. The algorithm of [6] is very similar; their contribution emphasizes the average time complexity analysis.

Yet another definition of cover of an indeterminate string was recently presented in [2]. They consider a rooted cover to be a prefix of the indeterminate text whose occurrences in the text cover all the positions of the text. Note that a rooted cover does not necessarily need to correspond to any solid cover; e.g., $\lozenge ab$ has a rooted cover of length 1 but no solid cover of length 1. An $O(n^2)$ worst-case time algorithm for computing all rooted covers of the text in the case of $\sigma = O(1)$ is presented in [2].

Our results: We show the following algorithms finding the length of a shortest solid cover of an indeterminate string:

1. an FPT algorithm parameterized by k and σ, with simple implementation and time complexity $O(nk + \sigma^{\lfloor k/2 \rfloor} k)$;
2. an FPT algorithm parameterized only by k, with time complexity $O(nk^2 + 2^k k^3)$;
3. for the case of partial words, a faster, $O(nk^2 + 2^{O(\sqrt{k}\log k)})$-time algorithm.

We also show hardness results, valid already for binary partial words:

1. NP-completeness of the covering problem;
2. that under the Exponential Time Hypothesis, no $2^{o(\sqrt{n})}$-time solution exists for the problem (this also rules out $2^{o(\sqrt{k})} n^{O(1)}$-time algorithms).

The algorithms are presented in Sections 3, 4, and 5, and the hardness results are given in Section 6.

A preliminary version of this article was presented as [12].

2. Preliminaries

An *indeterminate string* T of length $|T| = n$ over a finite alphabet Σ is a sequence $T[1]\dots T[n]$ such that for every index $i = 1, \dots, n$, the symbol $T[i]$ is a non-empty subset of Σ. By ε we denote the empty indeterminate string. If $|T[i]| = 1$, that is, $T[i]$ represents a single symbol of Σ, we say that $T[i]$ is a *solid* symbol. For convenience we often write that $T[i] = c$ instead of $T[i] = \{c\}$ in this case ($c \in \Sigma$). Otherwise, we say that $T[i]$ is a *non-solid* symbol. In what follows, by k we denote the number of non-solid symbols in the considered indeterminate string T and by σ we denote $|\Sigma|$. If $k = 0$, we call T a (solid) string.

We say that two indeterminate strings U and V *match* (denoted as $U \approx V$) if $|U| = |V|$ and for each $i = 1, \dots, |U|$ we have $U[i] \cap V[i] \neq \emptyset$. If $U \approx V$, we can define an indeterminate string $U \sqcap V$ with $(U \sqcap V)[i] = U[i] \cap V[i]$.

Example 1. Let

$$A = a\{b,c,d\}, \quad B = a\{a,b,d\}, \quad C = aa$$

be indeterminate strings (C is a solid string). Then:

$$A \approx B \quad (A \sqcap B = a\{b,d\}) \quad \text{and} \quad B \approx C \quad (B \sqcap C = aa), \quad \text{but} \quad A \not\approx C.$$

If all symbols $T[i]$ are either solid or equal to Σ, then T is called a *partial word*. In this case, the non-solid "don't care" symbol is denoted as $\lozenge$.

By $T[i..j]$ we denote a *factor* $T[i]\dots T[j]$ of T. If $i = 1$, the factor is called a *prefix* and if $j = n$, it is called a *suffix* of T. We say that an indeterminate pattern S *occurs* in an indeterminate text T at position j if $S \approx T[j..j+|S|-1]$. We define the *occurrence set* of S in T, denoted $Occ(S,T)$, as the set of all such positions j.

A *cover* of T is a solid string S such that each position i of T is covered by an occurrence of S in T, i.e., $Occ(S,T) \cap \{i - |S| + 1, \dots, i\} \neq \emptyset$ for every $i = 1, \dots, n$. If S is a cover of T, any subset $\mathcal{C} \subseteq Occ(S,T)$ already satisfying the latter property for all $i = 1, \dots, n$ is called a *covering set* of S; see Fig. 2.

Observation 1. Let $\mathcal{C}$ be a minimal covering set of a cover S of T. Then each position of T is covered by one or two occurrences $T[i..i+|S|-1]$ for $i \in \mathcal{C}$.

Remark 1. The shortest cover of an indeterminate string T need not be a cover of one of the solid strings matching T. For example, for a partial word $T = a\lozenge b$ over $\Sigma = \{a,b\}$, the shortest cover ab has length 2, whereas neither of the solid strings aab, abb has a cover of length 2.

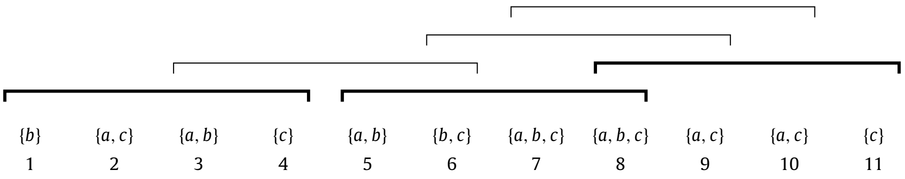


Fig. 2. An indeterminate string of length 11 with a cover *bcac*. All occurrences of the cover are marked; the smallest covering set is $\mathcal{C} = \{1, 5, 8\}$ (marked in bold).

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$T[i]$	$\Diamond$	a	$\Diamond$	a	b	a	$\Diamond$	a	b	a	b	a	a	$\Diamond$	a	$\Diamond$	a	b
$Pref[i]$	18	3	10	1	8	3	8	1	4	1	4	7	4	5	3	3	1	1
$\mathcal{F}[i]$	$\Diamond$	a	$\Diamond$	a	b	a	$\Diamond$	a	b	a	b	a	a	$\Diamond$	a	$\Diamond$	a	b
	$\Diamond$	a	b		$\Diamond$	a	b		b		a	$\Diamond$	a	$\Diamond$	a	b		
	$\Diamond$		b		b		a					b						
	$\Diamond$																	
	$\Diamond$																	

Fig. 3. The prefix table *Pref* and the fill-in table $\mathcal{F}$ for $T = \Diamond a \Diamond aba \Diamond ababaa \Diamond a \Diamond ab$. The fill-in sequences $\mathcal{F}[i]$ are shown vertically in the last row.

We say that S is a *solid prefix* of T if it is a solid string that matches the prefix $T[1..|S|]$. If $T[i]$ is solid for $i \le |S|$, then we must have $S[i] = T[i]$. Consequently, in order to specify S, it suffices to set the length $|S|$ and the characters $S[i]$ at positions $i \le |S|$ such that $T[i]$ is non-solid.

Assume that U and V are two matching indeterminate strings. Let $i_1, \dots, i_\ell$ be the positions of non-solid symbols in U. Then we define *fill-in*(U, V) as an indeterminate string F of length ℓ such that $F[j] = U[i_j] \cap V[i_j]$ for $j = 1, \dots, \ell$.

Example 2. *fill-in*$(a\Diamond a\Diamond a\Diamond\Diamond, \Diamond b\Diamond b\Diamond b\Diamond) = bbb\Diamond$, *fill-in*$(\Diamond b\Diamond b\Diamond b\Diamond, a\Diamond a\Diamond a\Diamond\Diamond) = aaa\Diamond$.

Given an indeterminate text T and indeterminate string U matching a prefix of T, we define the *fill-in sequence* of U as $F_U =$ *fill-in*$(T[1\,..\,|U|], U)$. If S is a solid prefix of T, then F_S and $|S|$ uniquely determine S.

2.1. Prefix table

The prefix table $Pref[1\,..\,n]$ of an indeterminate string T of length n stores at $Pref[i]$ the length of the longest matching prefix of T and $T[i\,..\,n]$. For convenience, we also compute the fill-in sequences of $T[1..Pref[i]] \sqcap T[i..i + Pref[i] - 1]$ and store them in a table $\mathcal{F}[1\,..\,n]$; see Fig. 3. Values $Pref[i]$ and $\mathcal{F}[i]$ let us easily characterize all solid prefixes occurring at position i:

Observation 2. For a solid prefix S of T, we have $i \in Occ(S, T)$ if and only if $Pref[i] \ge |S|$ and F_S matches a prefix of $\mathcal{F}[i]$. The latter condition can be tested in $O(k)$ time.

In the next lemma we show that the tables *Pref* and $\mathcal{F}$ can be computed efficiently. Our algorithm is an adaptation of the kangaroo method [14,25].

Lemma 1. *Given an indeterminate string T, the prefix table Pref and the fill-in table $\mathcal{F}$ can be computed in $O(nk + \sigma k^2)$ time.*

Proof. For convenience, we assume that the non-solid symbols in T are given as bit vectors (indexed with the alphabet) and the solid symbols have a constant-space representation. This way, $T[i] \cap T[j]$ can be computed in $O(1)$ time if at least one of the symbols is solid; otherwise, such an operation takes $O(\sigma)$ time.

A naive way to determine the value $Pref[i]$ is to compare $T[j]$ with $T[i + j - 1]$ for consecutive indices $j \ge 1$. This approach yields an $O(n^2 + \sigma k^2)$-time algorithm, since each pair of characters is compared at most once. Note that we spend most time matching solid characters. This process can be implemented more efficiently using LCP queries. Let us define a solid string $T_\$$ by substituting non-solid symbols in T with distinct characters $\$_1, \dots, \$_k \notin \Sigma$. Recall that $lcp_{T_\$}(i, j)$ is defined as the longest common prefix of $T_\$[i\,..\,n]$ and $T_\$[j\,..\,n]$. Moreover, the string $T_\$$ can be preprocessed in $O(n)$ time so that any $lcp_{T_\$}(i, j)$ query can be answered in $O(1)$ time; see [11].

Now, the values $Pref[i]$ and $\mathcal{F}[i]$ for $i > 1$ can be computed much more efficiently:

```
Input: Position $i > 1$
Output: Values $Pref[i]$ and $\mathcal{F}[i]$
$j := 1$;
$\mathcal{F}[i] = \varepsilon$;
while $i + j - 1 \le n$ and $T[i + j - 1] \approx T[j]$ do
    if $T[j]$ is non-solid then append $T[i + j - 1] \cap T[j]$ to $\mathcal{F}[i]$;
    $j := j + 1 + lcp_{T_\$}(i + j, j + 1)$;
$Pref[i] := j - 1$;
```

For each pair of symbols $T[i + j - 1]$, $T[j]$ compared in the while-loop, we have $j = 1$, $T[i + j - 1] \not\approx T[j]$ (in this case the loop terminates), or at least one of the symbols is non-solid. Hence, there are at most $2k + 1$ iterations for every i. No two symbols are compared twice, so the overall running time (across all positions i) is $O(nk + \sigma k^2)$. □

3. Naive algorithm parameterized by *k* and *σ*

We show the tools developed in Section 2 in action in a preliminary algorithm for the shortest cover problem.

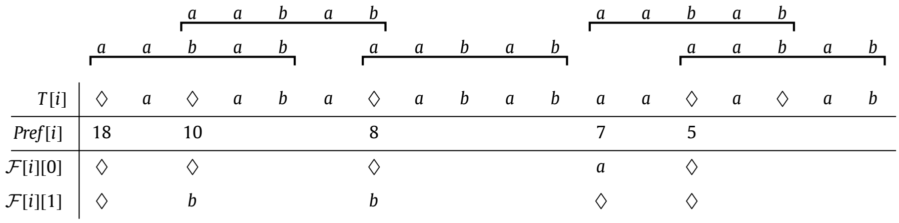
Observe that a cover S of T must be a solid prefix of T. Hence, its occurrence set $Occ(S, T)$ is characterized by Observation 2, and thus it can be computed in $O(nk)$ time using the prefix and fill-in tables. The following notion of *maxgap* is a standard tool allowing for an alternative definition of covers based on the occurrence sets.

For an increasing list of integers $L = [i_1, i_2, i_3, \dots, i_m]$ of length $m \ge 2$, we define

$$maxgap(L) = \max\{i_{t+1} - i_t \; : \; t = 1, \dots, m - 1\}.$$

Observation 3. A set $\mathcal{P} \subseteq Occ(S, T)$ is a covering set of a solid string S if and only if $1 \in \mathcal{P}$ and $maxgap(\mathcal{P} \cup \{n + 1\}) \le |S|$.

To test if S is a cover of T, it suffices to use Observation 3 for $\mathcal{P} = Occ(S, T)$; see Fig. 4. Combined with our initial observation, this yields the following corollary.



$T[i]$	◇	a	◇	a	b	a	◇	a	b	a	b	a	a	◇	a	◇	a	b
$Pref[i]$	18		10				8					7		5				
$\mathcal{F}[i][0]$	◇		◇				◇					a		◇				
$\mathcal{F}[i][1]$	◇		b				b					◇		◇				

Fig. 4. A solid string $S = aabab$ covering an indeterminate string $T = \Diamond a\Diamond aba\Diamond ababaa\Diamond a\Diamond ab$. We have $Occ(S, T) = \{1, 3, 7, 12, 14\}$ and $maxgap(Occ(S, T) \cup \{19\}) = 5 \le |S|$. As noted in Observation 2, positions $i \in Occ(S, T)$ can be characterized by $Pref[i] \ge 5$, $\mathcal{F}[i][0] \approx a$, and $\mathcal{F}[i][1] \approx b$.

Corollary 1. *Given an indeterminate string T along with its prefix table Pref and fill-in table $\mathcal{F}$, one can test in $O(nk)$ time if a given solid string S is a cover of T.*

The total number of solid prefixes is bounded by $n\sigma^k$ (recall that the length and the fill-in sequence represent a solid prefix uniquely).

Note that if a cover S satisfies $|S| \ge \frac{n}{2}$, then $\{1, n - |S| + 1\}$ is already a covering set. Thus, in order to check if T has a cover of length $m \ge \frac{n}{2}$, it suffices to verify if $Pref[n - m + 1] \ge m$. This lets us focus on covers of length at most $\lfloor \frac{n}{2} \rfloor$. Moreover, without loss of generality (by possibly reversing the input string T) we may assume that $T[1..\lfloor \frac{n}{2} \rfloor]$ contains at most $\left\lfloor \frac{k}{2} \right\rfloor$ non-solid symbols. This leaves us with $O(n\sigma^{\lfloor k/2 \rfloor})$ solid prefixes to test; see the pseudocode of *Prelim-Cover*.

Please cite this article in press as: M. Crochemore et al., Covering problems for partial words and for indeterminate strings, Theoret. Comput. Sci. (2017), http://dx.doi.org/10.1016/j.tcs.2017.05.026

```
Algorithm Prelim-Cover($T$)
    Input: An indeterminate string $T$ of length $n$ with $k$ non-solid symbols
    Output: The length of a shortest cover of $T$
    if $T[1..\lfloor n/2 \rfloor]$ contains more than $k/2$ non-solid symbols then Reverse $T$;
    Compute the tables $Pref$ and $\mathcal{F}$; { $O(nk^2)$ time, Lemma 1 }
    foreach solid prefix $S$ of $T$, $|S| \le n/2$, in non-decreasing length { $O(n\sigma^{\lfloor k/2 \rfloor})$ } do
        Compute $F_S$;
        { Observation 2, $O(nk)$ time }
        for $i := 1$ to $n$ do
            if $Pref[i] \ge |S|$ and $F_S \approx \mathcal{F}[i][1\,..\,|F_S|]$ then insert($Occ$, $i$);
        if $maxgap(Occ \cup \{n+1\}) \le |S|$ then return $|S|$;
    for $m := \lfloor n/2 \rfloor + 1$ to $n$ do
        if $Pref[n-m+1] \ge m$ then return $m$;
```

Proposition 1. *The shortest cover of an indeterminate string of length n containing k non-solid symbols can be computed in $O(n^2 k \sigma^{\lfloor k/2 \rfloor})$ time.*

4. Efficient algorithm parameterized by *k* and *σ*

We improve upon the result from the previous section in two steps.

4.1. First improvement

We will show that solid prefixes sharing the same fill-in sequence F can be tested together in $O(nk)$ time. We introduce a *ShortestCover* subroutine which, for a given string F, checks if there is a cover S of T with the fill-in sequence $F_S = F$. If so, the procedure returns the length of the shortest such cover.

Note that the lengths of solid prefixes with fill-in sequence F form an interval $[b, e]$, where b is the position of the $|F|$-th non-solid symbol in T and e is the position preceding the $(|F|+1)$-th non-solid symbol ($e = n$ if $|F| = k$). In particular, this interval can be retrieved efficiently if such positions are memorized.

Lemma 2. *The algorithm ShortestCover(F) correctly computes the shortest cover having the fill-in sequence F, if there is any.*

```
Algorithm ShortestCover($F$)
    Input: A string $F$ of length $|F| \le k$
    Output: The length of the shortest cover of $T$ with fill-in sequence $F$
PREPROCESSING:
    $[b, e] :=$ range of lengths of solid prefixes with fill-in sequence $F$;
    $L_F := \{i : F$ matches a prefix of $\mathcal{F}[i]\}$;
    $D_F := \{(Pref[i], i) : i \in L_F\}$;
PROCESSING:
    if $1 \notin L_F$ then return no solution;
    $M := L_F \cup \{n+1\}$;
    foreach $(p, i) \in D_F$ in increasing order do
        if $maxgap(M) \le \min(p, e)$ then return $\max(b, maxgap(M))$;
        remove $i$ from $M$;
    return no solution;
```

Proof. Due to Observation 2, for every string S such that $F_S = F$, we have $Occ(S, T) = \{i \in L_F : Pref[i] \ge |S|\}$. Hence, while processing the first pair $(Pref[i], i) \in D_F$ with $Pref[i] \ge |S|$, we have $M = Occ(S, T) \cup \{n+1\}$.

We check if there is a solid prefix S of T having the fill-in sequence F in the first line of the processing phase (this could not be the case if T is not a partial word and some $F[i]$ does not match the respective non-solid symbol in T). If so, then, by Observation 3, S is a cover of T if and only if $maxgap(M) \leq |S|$.

When processing $(Pref[i], i) \in D_F$ we do not know $|S|$, but we know that it belongs to the interval $[b, \min(p, e)]$. This implies the condition for computing $|S|$ that is used in the pseudocode. □

We can also note that the computations in the pseudocode could be ended (with a negative answer) after the first pair $(p, i) \in D_F$ with $p \geq e$ has been processed.

Lemma 3. *The algorithm ShortestCover*(F) *works in* $O(nk)$ *time assuming that the prefix table Pref and the fill-in table* $\mathcal{F}$ *are available.*

Proof. In the preprocessing phase, the (sorted) list L_F can be computed in $O(nk)$ time and the set D_F can be computed and sorted in $O(n)$ time (e.g., using bucket sort).

In the processing phase, we store an array of pointers mapping each $i \in M$ to the corresponding element of the list M. Thanks to it, each $i \in M$ can be removed from M in constant time. Observe that $maxgap(M)$ may only increase and it can be updated in constant time subject to deletion of elements from M. Also note that $n+1$ is never deleted from M and 1 may only be deleted in the last iteration, so $|M| \geq 2$ each time we retrieve $maxgap(M)$. □

The number of potential fill-in sequences of length $\leq \frac{k}{2}$ that we need to test is $\sum_{\ell=0}^{\lfloor k/2 \rfloor} \sigma^{\ell} = O(\sigma^{\lfloor k/2 \rfloor})$, so the overall running time is $O(nk\sigma^{\lfloor k/2 \rfloor})$.

4.2. Second improvement

The second improvement is based on the fact that the complexity of the function *ShortestCover*(F) is dominated by preprocessing; the processing phase takes only $O(|L_F|)$ time. Let us denote by *ShortestCover*(F, L_F, D_F) the sole processing phase of the routine. Observe that having constructed the list L_F for a fill-in sequence F, we may construct the lists L_{Fc} for all extensions of F by a character c; see Fig. 5. This requires a single scan over the list L_F: we insert each $i \in L_F$ to the list L_{Fc} for every $c \in \Sigma$ matching $\mathcal{F}[i][|F|+1]$. The lists D_{Fc} can be constructed in the same way; cf. the pseudocode of the *ShortestCoverRec* routine.

```
Algorithm ShortestCoverRec($F, L_F, D_F$)
    Input: A string $F$ of length $|F| \leq k/2$ together with $L_F$ and $D_F$
    Output: The length of the shortest cover of $T$ with fill-in sequence of length
            $\leq k/2$ having a prefix $F$
    $m :=$ ShortestCover($F, L_F, D_F$);
    if $|F| + 1 > k/2$ then return $m$;
    foreach $i \in L_F$ do
        foreach $c \in \mathcal{F}[i][|F|+1]$ do  insert($L[c], i$);
    foreach $(p, i) \in D_F$ do
        foreach $c \in \mathcal{F}[i][|F|+1]$ do  insert($D[c], (p, i)$);
    foreach $c \in \Sigma$ do
        if $L[c] \neq \emptyset$ then  $m := \min(m,$ ShortestCoverRec($Fc, L[c], D[c]$));
    return $m$;
```

The array of pointers used to represent M can be constructed in $O(|L_F|)$ time using $O(n)$ space common to all *ShortestCover* calls. The overall time complexity of *ShortestCoverRec* is proportional to the total size of lists L_F across all fill-in sequences F plus σ times the total number of recursive calls with $|F|+1 \leq k/2$. The latter term is $O(\sigma^{\lfloor k/2 \rfloor})$. To bound the former, note that each position i may occur in lists L_F for at most $\sigma^{k_{i,f}}$ fill-in sequences F of length f, where $k_{i,f}$ is the number of non-solid positions within $\mathcal{F}[i][1 \mathinner{..} f]$. Moreover, we have $\sum_{i=1}^{n} k_{i,f} \leq kf$, because for any fixed j the symbol $\mathcal{F}[i][j]$ can be non-solid for at most k positions i. Since $0 \leq k_{i,f} \leq f$, convexity of the exponential function yields $\sum_{i=1}^{n} \sigma^{k_{i,f}} \leq n - k + k\sigma^{f} \leq n + k\sigma^{f}$ (formally, this is due to Karamata's inequality). Summing up over all lengths f $(0 \leq f \leq \lfloor k/2 \rfloor)$, we get a bound of $O(nk + k\sigma^{\lfloor k/2 \rfloor})$ on the overall running time taken by *ShortestCover*(F) calls. By Lemma 1, the prefix and fill-in tables can be constructed in $O(nk + \sigma k^2)$ time, which is dominated by $O(nk + k\sigma^{\lfloor k/2 \rfloor})$. Consequently, we arrive at the following result:

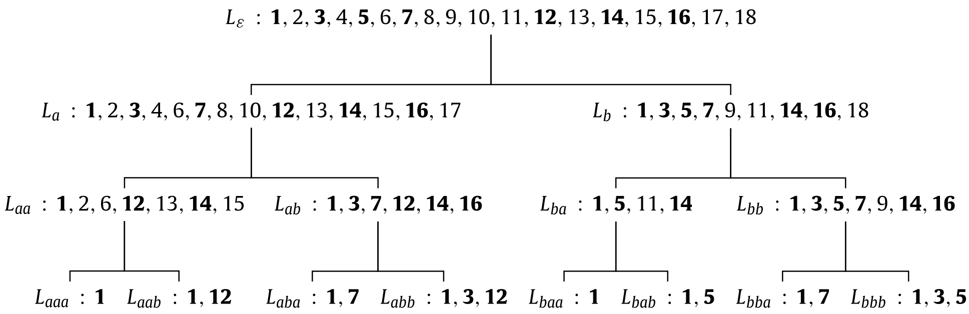

L_ε : **1**, 2, **3**, 4, **5**, 6, **7**, 8, 9, 10, 11, **12**, 13, **14**, 15, **16**, 17, 18
L_a : **1**, 2, **3**, 4, 6, **7**, 8, 10, **12**, 13, **14**, 15, **16**, 17
L_b : **1**, **3**, **5**, **7**, 9, 11, **14**, **16**, 18
L_{aa} : **1**, 2, 6, **12**, 13, **14**, 15
L_{ab} : **1**, **3**, **7**, **12**, **14**, **16**
L_{ba} : **1**, **5**, 11, **14**
L_{bb} : **1**, **3**, **5**, **7**, 9, **14**, **16**
L_{aaa} : **1**
L_{aab} : **1**, **12**
L_{aba} : **1**, **7**
L_{abb} : **1**, **3**, **12**
L_{baa} : **1**
L_{bab} : **1**, **5**
L_{bba} : **1**, **7**
L_{bbb} : **1**, **3**, **5**

Fig. 5. Lists L_F for text $T = \lozenge a\lozenge aba\lozenge ababaa\lozenge a\lozenge ab$ and all binary fill-in sequences F of length ≤ 3. Ambiguous positions (see Section 5) are marked in bold.

Theorem 1. *The shortest cover of an indeterminate string with k non-solid symbols can be computed in $O(nk + k\sigma^{\lfloor k/2 \rfloor})$ time.*

5. Algorithm parameterized by *k*

Note that in the running time of Theorem 1, the $O(k\sigma^{\lfloor k/2 \rfloor})$ term is a contribution of positions i such that $\mathcal{F}[i]$ contains a non-solid symbol, as each position i with a solid $\mathcal{F}[i]$ can occur in up to $|\mathcal{F}[i]|$ lists L_F (cf. Fig. 5). We call the former positions *ambiguous*, while the latter positions are *unambiguous*. We shall denote the set of ambiguous positions by $\mathcal{A}$.

Observation 4. $|\mathcal{A}| = O(k^2)$.

We say that a solid prefix S is a *solid-match-prefix* of T if F_S is *equal to* a prefix of $\mathcal{F}[i]$ for some position i. The following observation is an important tool in our algorithms.

Observation 5. If a solid prefix S is *not* a solid-match-prefix, then $Occ(S, T) \subseteq \mathcal{A}$.

Example 3. Consider $T = \lozenge a\lozenge aba\lozenge ababaa\lozenge a\lozenge ab$ (see Figs. 3 and 5); its ambiguous positions are $1, 3, 5, 7, 12, 14, 16$. The string aaa is a solid-match-prefix of T, as $F_{aaa} = aa$ is equal to a prefix of $\mathcal{F}[2]$. On the other hand, aab is a solid prefix of T but not a solid-match-prefix of T; it occurs at positions $L_{ab} = 1, 3, 7, 12, 14, 16$, all of which are ambiguous.

We use different tools to find covers of T which are solid-match-prefixes of T and those which are not. Dealing with solid-match-prefixes is easier: as there are $O(nk)$ prefixes F of fill-in sequences $\mathcal{F}[i]$ across positions i, it is straightforward to devise an $O(n^2k^2)$-time algorithm using *ShortestCover* procedure for each F. Below we present an $O(nk^2)$-time solution based on techniques developed for Theorem 1.

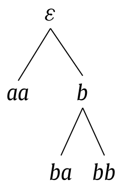
Theorem 2. *The shortest cover among all solid-match-prefixes can be computed in $O(nk^2)$ time.*

Proof. Let $\mathbf{F}$ be the family of fill-in sequences that need to be processed. Formally,

$$\mathbf{F} = \{\mathcal{F}[i][1\mathinner{.\,.} j] : 0 \le j \le |\mathcal{F}[i]|, \mathcal{F}[i][1\mathinner{.\,.} j] \text{ is solid}\}.$$

Note that $|\mathbf{F}| = O(nk)$ and $\sum_{F\in\mathbf{F}} |F| = O(nk^2)$. For each $F \in \mathbf{F}$ let us denote the set of possible *extensions*: $\text{ext}(F) = \{c \in \Sigma : Fc \in \mathbf{F}\}$. We say that F is *branching* if $|\text{ext}(F)| \ge 2$ and *non-extendible* if $\text{ext}(F) = \emptyset$. Note that there are at most n non-extendible elements in $\mathbf{F}$. Hence, the set $\mathbf{F}$ can be viewed as a compacted TRIE with at most n leaves (non-extendible strings), at most $n - 1$ branching nodes (branching strings) and at most n compacted edges (this is the sum of $\text{ext}(F)$ over all branching strings F).

Example 4. For $T = \lozenge a\lozenge aba\lozenge ababaa\lozenge a\lozenge ab$ (see Fig. 3), $\mathbf{F} = \{\varepsilon, a, b, aa, ba, bb\}$. The branching strings are ε and b, and the non-extendible strings are aa, ba, and bb.



Please cite this article in press as: M. Crochemore et al., Covering problems for partial words and for indeterminate strings, Theoret. Comput. Sci. (2017), http://dx.doi.org/10.1016/j.tcs.2017.05.026

We apply procedure *ShortestCover* for each $F \in \mathbf{F}$. We follow the lines of *ShortestCoverRec* to construct the lists L_F (and D_F) for $F \in \mathbf{F}$. The only difference in the implementation is that, when checking if a given element $i \in L_F$ should be inserted into L_{Fc}, we only consider $c \in \text{ext}(F)$. Our goal now is to bound the total size of L_F across $F \in \mathbf{F}$.

If $\mathcal{F}[i][|F|+1]$ is non-solid and F is branching, we process such $i \in L_F$ in time $O(|\text{ext}(F)|)$, iterating over all characters $c \in \text{ext}(F)$. As we already noticed, the sum of $|\text{ext}(F)|$ across all branching fill-in sequences F is at most n, and, for a given F, $\mathcal{F}[i][|F|+1]$ can be non-solid for at most k positions i. Consequently, this case contributes $O(nk)$ to the total running time. In particular, the total number of elements inserted this way to the lists L_F is also $O(nk)$.

Otherwise, we have just one character c to consider and we process such $i \in L_F$ in $O(1)$ time, possibly inserting a *copy* of i to L_{Fc}. This way, for each of the n elements originally introduced in L_ε and for each of $O(nk)$ elements introduced using the previous case we create a chain of up to k such copies. Hence, the total size of the lists L_F is $O(nk^2)$ and so is the total running time of the *ShortestCover* instances. Apart from that, we only use Lemma 1, whose running time $O(nk+\sigma k^2)$ is dominated by $O(nk^2)$. □

The key property of solid-match-prefixes S was that S could be determined by $|S|$ and the fill-in sequence $\mathcal{F}[i]$ at a single position i. In general, in case of a partial word it suffices to use up to $2\sqrt{k}$ positions to retrieve all solid characters of the cover. This fact can be stated generally for any indeterminate string as follows.

Lemma 4. *Let S be a cover of an indeterminate string T and let $\mathcal{C}$ be its minimal covering set. There exists a subset $\mathcal{R}$ of $\mathcal{C}$ of size at most $2\sqrt{k}$ such that, for every position j of the fill-in sequence F_S, if $F_S[j] = \mathcal{F}[i][j]$ for some $i \in \mathcal{C}$, then $F_S[j] = \mathcal{F}[i][j]$ for some $i \in \mathcal{R}$.*

Proof. We shall prove that any minimal subset $\mathcal{R}$ satisfying the desired property is of size at most $2\sqrt{k}$. Since $\mathcal{R}$ is minimal, for every $i \in \mathcal{R}$ there must be a position j which prevents the removal of i from $\mathcal{R}$, i.e., such that $\mathcal{F}[i'][j]$ is non-solid for every $i' \in \mathcal{R}$, $i' \neq i$. This means that each $i \in \mathcal{R}$ implies $r-1$ non-solid positions in T that correspond to $\mathcal{F}[i'][j]$ for $i' \in \mathcal{R}$ and some $1 \le j \le |F_S|$. This is at least $r \cdot (r-1)$ non-solid positions in total, where $r = |\mathcal{R}|$.

Some of these $r \cdot (r-1)$ non-solid positions may be the same in T. However, by Observation 1, any non-solid position is covered by at most two occurrences of S in $\mathcal{C}$. Therefore, $r \cdot (r-1) \le 2k$ and, consequently, $r \le 2\sqrt{k}$. □

For a set of positions $\mathcal{P}$, we introduce an auxiliary operation *TestCover*$(\mathcal{P})$ which checks if there is a cover of T for which $\mathcal{P}$ is a covering set and, if so, retrieves such a cover. Note that the length of such a cover is fixed to $n+1-\max\mathcal{P}$.

Lemma 5. *After $O((n+2^k)k^2)$-time preprocessing, TestCover$(\mathcal{P})$ can be implemented in $O(|\mathcal{P}|k)$ time.*

Proof. Let $m = n+1-\max\mathcal{P}$. By Observation 3, $\mathcal{P}$ can be a covering set for a cover of length m only if $1 \in \mathcal{P}$ and $\mathit{maxgap}(\mathcal{P} \cup \{n+1\}) \le m$. These conditions can be easily checked in $O(|\mathcal{P}|)$ time without any preprocessing.

Now, it suffices to check if there is a solid string S of length m such that $T[i..i+m-1] \approx S$ for all $i \in \mathcal{P}$. Such a string certainly does not exist if $\mathit{Pref}[i] < m$ for some $i \in \mathcal{P}$. Otherwise, let the set Y contain positions of all don't care symbols in $T[1..m]$. We need to check, for each $j \in Y$, how many solid symbols does the set

$$X_j = \{T[i-1+j] : i \in \mathcal{P}\}$$

contain. If there are two different solid symbols in this set, then there is no such cover. If there is exactly one symbol, then it suffices to check in $O(|\mathcal{P}|)$ time if this symbol matches all the non-solid symbols in this set. Otherwise, if there are no such solid symbols, we will retrieve the result for such X_j from the results of preprocessing. The processing phase takes $O(|\mathcal{P}|k)$ time, as there are $|Y| \le k$ positions j to test.

The preprocessing phase starts with computing *Pref* table using Lemma 1 in $O(nk^2)$ time. Let Z be the set of all non-solid positions in T. We wish to compute, for each subset $X \subseteq Z$, a Boolean value stating if there is a single solid symbol matching all the positions in X. First, for every character $c \in \Sigma$, we compute a subset $Z_c \subseteq Z$ of non-solid positions containing c. This takes $O(\sigma k)$ time. Initially, we set the Boolean values of the sets Z_c to 'true' and the remaining values to 'false'. Finally, we scan all subsets of Z in a non-increasing order of sizes to compute the final answer, that is, whether $X \subseteq Z_c$ for some $c \in \Sigma$. To process X, it suffices to 'or' the value at X with each of the values at $X \setminus \{x\}$ for each $x \in X$. Altogether, this preprocessing takes $O(2^k \cdot k^2)$ time. □

We apply the *TestCover* routine to obtain an efficient solution for covering an indeterminate string with non-solid-match prefixes, hence, for the main problem. The algorithm of Theorem 3 is summarized in the pseudocode of the *FastCover* algorithm.

Theorem 3. *The shortest cover of an indeterminate string T of length n with k non-solid symbols can be computed in $O(nk^2 + 2^k k^3)$ time.*

Please cite this article in press as: M. Crochemore et al., Covering problems for partial words and for indeterminate strings, Theoret. Comput. Sci. (2017), http://dx.doi.org/10.1016/j.tcs.2017.05.026

Proof. Let S be a shortest cover of the given indeterminate string T, let m be its length, and let $\mathcal{C}$ be its minimal covering set.

By Theorem 2, if S is a solid-match-prefix, then it can be computed in $O(nk^2)$ time. Hence, we may assume that S is not a solid-match-prefix. By Observation 5 this means that $\mathcal{C} \subseteq \mathcal{A}$. A straightforward approach, leading to an $O((n+2^{k^2})k^2)$-time algorithm, would be to apply the *TestCover* procedure for all subsets of $\mathcal{A}$. Below, we develop a much more efficient solution, which is based on a distinction into two cases:

Case 1: There exists an index $j \in \{1, \ldots, |F_S|\}$ such that $\mathcal{F}[i][j]$ is non-solid for every $i \in \mathcal{C}$. We have $O(k \cdot 2^k)$ possible covering sets $\mathcal{C}$: there are k possibilities for j, and at most 2^k possible subsets of $\{i' : T[i'][j] \text{ is non-solid}\}$. Each such set $\mathcal{C}$ has size at most k. Using the *TestCover* routine, we process these sets in $O(k^3 \cdot 2^k)$ total time.

```
Algorithm FastCover($T$)
    Input: An indeterminate string $T$ of length $n$ with $k$ non-solid symbols
    Output: The length of the shortest cover of $T$
    $result$ := the length of the shortest solid-match-prefix cover of $T$; { $O(nk^2)$ time }
  { Implementation of Case 1 }
    for $j := 1$ to $k$ do
        foreach $\mathcal{C} \subseteq \{i' : T[i'][j]$ is non-solid$\}$ do
            if TestCover($\mathcal{C}$) then
                $result := \min(result, n + 1 - \max \mathcal{C})$;
  { Implementation of Case 2 }
    foreach $m \in \{n + 1 - a : a \in \mathcal{A}\}$ do
        $f$ := the number of non-solid symbols in $T[1..m]$;
        foreach $\mathcal{R} \subseteq \mathcal{A}, |\mathcal{R}| \le 2\sqrt{k}$ do
            if there exists $i \in \mathcal{R}$ such that $Pref[i] < m$ then continue;
            for $j := 1$ to $f$ do
                $X := \{\mathcal{F}[i][j] : i \in \mathcal{R}\} \cap \Sigma$;
                $F[j]$ := any element of $X$ or any element of $\Sigma$, if $X = \emptyset$;
            $\mathcal{P} := \emptyset$;
            foreach $i \in \mathcal{A}$ do
                if $Pref[i] \ge m$ and $F \approx \mathcal{F}[i][1..f]$ then insert($\mathcal{P}, i$);
            if TestCover($\mathcal{P}$) then
                $result := \min(result, m)$;
    return $result$;
```

Case 2: For every $j \in \{1, \ldots, |F_S|\}$ there exists an index $i \in \mathcal{C}$ such that $\mathcal{F}[i][j]$ is solid. By Lemma 4 there is a subset $\mathcal{R} \subseteq \mathcal{C}$ of size at most $2\sqrt{k}$ such that for every $j \in \{1, \ldots, |F_S|\}$ there exists an index $i \in \mathcal{R}$ such that $\mathcal{F}[i][j]$ is solid. Note that, for a fixed length m, given such a subset $\mathcal{R}$ we can uniquely retrieve F_S (and S). Moreover, $Occ(S, T) \cap \mathcal{A}$ must be a covering set of S (we do not require to compute $\mathcal{C}$).

This leads to the following algorithm resulting in $k^2 \cdot 2^{O(\sqrt{k}\log k)}$ sets to test. First, we have $|\mathcal{A}| \le k^2$ possibilities for the length of the cover m (since $n - m + 1 \in \mathcal{C} \subseteq \mathcal{A}$). Next, we generate all subsets of $\mathcal{A}$ of size at most $2\sqrt{k}$. The number of such sets is:

$$\sum_{p=1}^{\lfloor 2\sqrt{k} \rfloor} \binom{|\mathcal{A}|}{p} \le \sum_{p=1}^{\lfloor 2\sqrt{k} \rfloor} \binom{k^2}{p} \le \sum_{p=1}^{\lfloor 2\sqrt{k} \rfloor} k^{2p} \le 2\sqrt{k} \cdot k^{2\sqrt{k}} = 2^{O(\sqrt{k}\log k)}.$$

For every generated set $\mathcal{R}$ we check if $Pref[i] \ge m$ for each $i \in \mathcal{R}$ and, for every $j \in \{1, \ldots, |F_S|\}$, we verify whether $\{\mathcal{F}[i][j] : i \in \mathcal{R}\}$ contains exactly one solid symbol. If so, we construct the fill-in sequence F_S by taking the unique symbol as $F_S[j]$. Otherwise, we may reject $\mathcal{R}$. (In the pseudocode, for simplicity, we select an arbitrary letter of Σ in this case.)

Finally, we use Observation 2 to check if $Occ(S, T) \cap \mathcal{A}$ is a covering set of S. Note that we may assume $Occ(S, T) \subseteq \mathcal{A}$, so this takes $O(k^3)$ time. The total running time in this case is $O(k^5 2^{O(\sqrt{k}\log k)}) = 2^{O(\sqrt{k}\log k)}$. □

We conclude with an algorithm for partial words which is faster than the generic solution for indeterminate strings.

Theorem 4. *The shortest cover of a partial word of length n with k don't care symbols can be computed in* $2^{O(\sqrt{k}\log k)} + O(nk^2)$ *time.*

Proof. Let S be a shortest cover of a given partial word T, let m be its length, and let $\mathcal{C}$ be its minimal covering set. As in the proof of Theorem 3, application of Theorem 2 lets us assume that $\mathcal{C} \subseteq \mathcal{A}$.

By Lemma 4 there is a set $\mathcal{R} \subseteq \mathcal{C}$ of size at most $2\sqrt{k}$ such that for every position j in F_S we either have $\mathcal{F}[i][j] = \lozenge$ for each $i \in \mathcal{C}$ or $F_S[j] = \mathcal{F}[i][j]$ for some $i \in \mathcal{R}$. Given the length m and the set $\mathcal{R}$, we may retrieve $F_S[j]$ for the latter positions, while for the former ones substituting $F_S[j]$ with an arbitrary character yields a solid prefix S' which is still a cover with covering set $\mathcal{C}$. Moreover, $\mathcal{C} \subseteq Occ(S', T) \cap \mathcal{A}$ and the latter is also a covering set of S'.

Consequently, we obtain an algorithm very similar to that used in Case 2 in the proof of Theorem 3: we have $O(k^2)$ possibilities for m, $2^{O(\sqrt{k}\log k)}$ possibilities for $\mathcal{R}$, and we verify each in $O(k^3)$ time. The overall running time is $2^{O(\sqrt{k}\log k)} + O(nk^2)$. $\square$

6. Hardness results

Hardness results obtained for partial words remain valid in the more general setting of the indeterminate strings, so in this section we restrict our considerations to partial words. We consider the following decision problem.

Problem 1 *(Shortest Cover in Partial Words).* Given a partial word T of length n over an alphabet Σ and an integer d, decide whether T has a solid cover of length at most d.

We devise a reduction from the CNF-SAT Problem. Recall that in this problem we are given a Boolean formula with p variables which is a conjunction of m clauses $C_1 \wedge C_2 \wedge \ldots \wedge C_m$, where each clause C_i is a disjunction of (positive or negative) literals, and our goal is to check if there exists an interpretation that satisfies the formula. Below we present a reformulation of the CNF-SAT Problem which is more suitable for our proof.

Problem 2 *(Universal Mismatch).* Given a collection of m binary partial words $W_1, \ldots, W_m$ each of length p, check if there exists a binary partial word V of length p such that $V \not\approx W_i$ for any i.

Observation 6. Given an instance of the CNF-SAT Problem with p variables and m clauses, in linear time one can construct an equivalent instance of the Universal Mismatch Problem with m partial words each of length p. The resulting mapping of instances is bijective and its inverse can also be computed in linear time.

Example 5. Consider a formula

$$\phi = (x_1 \vee x_2 \vee \neg x_3 \vee x_5) \wedge (\neg x_1 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_5)$$

with three clauses and five variables. In the corresponding instance of the Universal Mismatch Problem, for each clause C_i we construct a partial word W_i such that $W_i[j] = 0$ if $x_j \in C_i$, $W_i[j] = 1$ if $\neg x_j \in C_i$, and $W_i[j] = \lozenge$ otherwise:

$$W_1 = 001\lozenge 0, \quad W_2 = 1\lozenge\lozenge 0\lozenge, \quad W_3 = \lozenge 10\lozenge 1.$$

The interpretations $(1, 0, 1, 1, 0)$, $(1, 1, 1, 1, 0)$ satisfy ϕ. They correspond to partial words 10110, 11110 and $1\lozenge 110$, none of which matches any of the partial words W_1, W_2, W_3.

Consider an instance $\mathbf{W} = (W_1, \ldots, W_m)$, $|W_j| = p$, of the Universal Mismatch Problem. We construct a binary partial word T of length $O(p(p+m))$ which is equivalent to $\mathbf{W}$ as an instance of the Shortest Cover in Partial Words Problem with $d = 4p + 3$.

We define a morphism

$$h: \quad 0 \to 0100, \quad 1 \to 0001, \quad \lozenge \to 0000.$$

Structure of the NP-hardness proof. We construct T so that a partial word V of length p is a solution to $\mathbf{W}$ if and only if $S = 11h(V)0$ covers T. The word T is of the form $11\pi^p 0\beta_1 \ldots \beta_p \gamma_{W_1} \ldots \gamma_{W_m}$, where $\pi = 0\lozenge 0\lozenge$ and β_j, γ_W are *gadgets* to be specified later. These gadgets are chosen so that every cover of T has length at least d and every d-cover of T (i.e., every cover of T of length exactly d) is a d-cover of each gadget string β_j and γ_W. Here, the prefix $11\pi^p 0$ and all β_j are *consistency* gadgets which guarantee that any d-cover is of the form $11h(V)0$ for some partial word V of length p. On the other hand, γ_W are *constraint* gadgets which do not allow V to match W.

6.1. Consistency gadgets

The prefix $11\pi^p 0$ of T enforces that any d-cover S of T is of the form $S = 11s_1 \ldots s_p 0$ where $s_j \approx \pi$ for each j. Thus, in order to make sure that S is of the form $11h(V)0$ for some partial word V, it suffices to rule out the possibility that $s_j = 0101$ for some j. To this end, we define

$$\beta_j = 11\,\pi^{p-1}\,0\,\lozenge^{4j+1}\,000\,\lozenge^d.$$

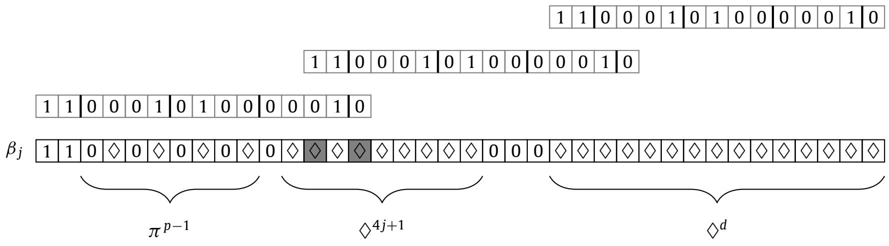


Fig. 6. Sample gadget β_j for $j=2$ and $p=3$ with occurrences of a pattern $11h(101)0$. Positions $d-2$ and d are marked in gray.

Observation 7. Suppose S is a solid string such that $S \approx 11\pi^p 0$. Then S occurs as a prefix and as a suffix of β_j.

Lemma 6. *Let $S = 11s_1 \dots s_p 0$ be a solid string with $s_i \approx \pi$ for each i. Then S covers β_j if and only if $s_j \neq 0101$.*

Proof. ($\Leftarrow$) By Observation 7, S occurs in β_j at positions 1 and $|\beta_j| - |S| + 1 = d + 4j + 1$. If $s_j \neq 0101$, then $s_j = 0100$ and S also occurs at position $d-2$, or $s_j = 0001$ and S occurs at position d, or $s_j = 0000$ and S occurs at both positions $d-2$ and d; see Fig. 6. Consequently, S covers β_j since

$$maxgap(1, d-2, d+4j+1) \leq d \quad \text{and} \quad maxgap(1, d, d+4j+1) \leq d.$$

($\Rightarrow$) If S covers β_j, it must have an occurrence at some position q with $2 \leq q \leq d+1$. In particular, 11 must occur at position q, which further restricts $q \in \{d-3, d-2, d-1, d, d+1\}$. If $s_j = 0101$, then we would need to have $\beta_j[q+4j-1] \approx 1$ and $\beta_j[q+4j+1] \approx 1$; see Fig. 6. However, $\beta_j[d+4j-2] = \beta_j[d+4j-1] = \beta_j[d+4j] = 0$. We get a contradiction for each of the five possible values of q. Consequently, S cannot have $s_j = 0101$. □

Corollary 2. *A solid string $S \approx 11\pi^p 0$ is a cover of each partial word β_j for $j = 1, \dots, p$ if and only if $S = 11h(V)0$ for a binary partial word V of length p.*

6.2. Constraint gadgets

We encode a constraint $V \not\approx W$ using a gadget

$$\gamma_W = 11\mu(W^R)010\Diamond^d$$

where W^R denotes the reverse of W and μ is the following morphism:

$$\mu: \quad 0 \to \Diamond\Diamond 0 \Diamond, \quad 1 \to 0\Diamond\Diamond\Diamond, \quad \Diamond \to 0\Diamond 0 \Diamond.$$

Observation 8. Suppose S is a solid string such that $S \approx 11\pi^p 0$ and W is a partial word of length p. Then S occurs as a prefix and as a suffix of γ_W.

Before we proceed with a proof that γ_W indeed encodes the constraint, let us characterize the relation between morphisms μ and h.

Lemma 7. *Let $c, c' \in \{0, 1, \Diamond\}$, and let X, Y be partial words of the same length. Then $11h(Xc)0$ occurs in $\mu(c'Y)\,010\,\Diamond\Diamond$ if and only if $c \not\approx c'$.*

Proof. Let $P = 11h(Xc)0$, $Q = \mu(c'Y)010\Diamond\Diamond$ and $\ell = |P|$.

($\Rightarrow$) Note that $|Q| = \ell + 2$, so P can occur in Q only at positions $p \in \{1, 2, 3\}$. Moreover, $p = 2$ is impossible because $Q[\ell - 1] = 1$ and $P[\ell - 2] = 0$ (since $h(c) \approx \pi = 0\Diamond 0 \Diamond$); see Fig. 7. Thus, P can occur in Q only as a prefix or as a suffix.

Suppose P occurs as a prefix of Q. Note that P begins with 11, so $\mu(c') \approx 11\Diamond\Diamond$ and thus $c' = 0$. Moreover, Q ends with $010\Diamond\Diamond$, so $h(c) \approx \Diamond\Diamond 01$ and $c = 1$. Similarly, if P occurs as a suffix of Q, then $\mu(c') \approx \Diamond\Diamond 11$, so $c' = 1$, and $h(c) \approx 010\Diamond$, so $c = 0$. Consequently, $c \not\approx c'$ in either case.

($\Leftarrow$) Observe that $\mu(c'Y)$ has $\Diamond$'s at all even positions, and $\Diamond$'s or zeroes at all odd positions, while, $h(Xc)0$ has zeroes at all odd positions. Thus, any mismatch preventing an occurrence of P as a prefix or as a suffix of Q must be due to the initial 11 in P or the terminal $010\Diamond\Diamond$ in Q. The corresponding positions in Q and P depend only on c' and c, respectively. As $c \not\approx c'$, we have $c = 1$ and $c' = 0$ or $c = 0$ and $c' = 1$. In the former case P occurs in Q as a prefix, and in the latter it occurs as a suffix; see Fig. 7. □

Lemma 8. *Let V and W be binary partial words of length p. Then $S = 11h(V)0$ covers γ_W if and only if $V \not\approx W$.*

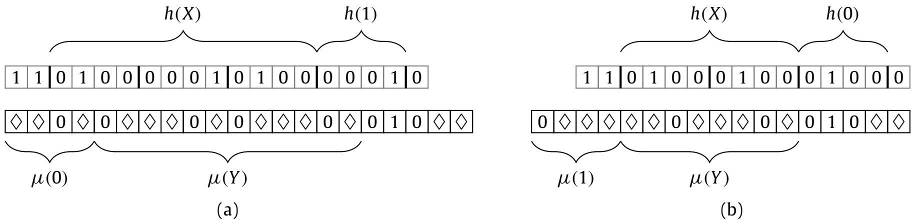


Fig. 7. Illustration of Lemma 7: an occurrence of $11h(Xc)0$ in $\mu(c'Y)010\Diamond\Diamond$ for (a) $Xc = 0101$, $c'Y = 01\Diamond 0$; (b) $Xc = 000$, $c'Y = 100$. In general, $11h(Xc)0$ is a prefix of $\mu(c'Y)010\Diamond\Diamond$ if $c = 1$ and $c' = 0$, and a suffix – if $c = 0$ and $c' = 1$.

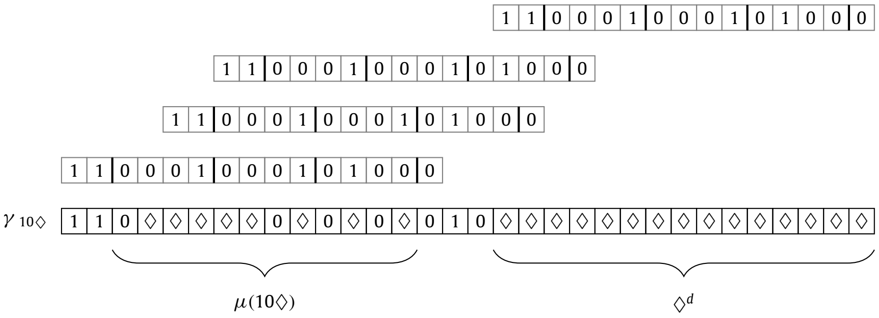


Fig. 8. A gadget $\gamma_{\Diamond 01}$ with occurrences of a pattern $11h(110)0$.

Proof. ($\Leftarrow$) Note that, by Observation 8, S always matches both a prefix and a suffix of γ_W. The only positions which are not covered by these two occurrences of S form the middle 10 factor $\gamma_W[d+1..d+2]$; see Fig. 8. If $V \not\approx W$, there exists a position $i \in \{1, \dots, p\}$ such that $V[i] \not\approx W[i]$. By Lemma 7, $11h(V[1..i])0$ occurs in $\mu((W[1..i])^R)010\Diamond\Diamond$. This occurrence extends to an occurrence of $11h(V)0$ in $\mu((W[1..i])^R)010\Diamond^{d-4i}$, and consequently an occurrence of $11h(V)0$ in γ_W covering the middle 10 factor $\gamma_W[d+1..d+2]$. Thus, $S = 11h(V)0$ is a cover of γ_W.

($\Rightarrow$) Let r be the position in γ_W corresponding to an occurrence of $11h(V)0$ that covers $\gamma_W[d+2]$. Note that S begins with 11, so $r < d-1$. Let $i = \lceil \frac{d-r}{4} \rceil$, i.e., i is the smallest value such that the occurrence of $11h(V[1..i])0$ at position r covers the middle 10 factor $\gamma_W[d+1..d+2]$. Now, observe that $11h(V[1..i])0$ occurs in $\mu((W[1..i])^R)010\Diamond\Diamond$, so Lemma 7 implies that $V[i] \not\approx W[i]$, and thus $V \not\approx W$. □

6.3. Main hardness results

Lemma 9. *Given an instance* **W** *of the* Universal Mismatch *Problem with m partial words of length p, one compute in $O(|T|)$ time a binary partial word T of length $\Theta((p+m)^2)$ for which the* Shortest Cover in Partial Words *Problem with $d = 4p+3$ is equivalent to* **W**.

Proof. Let

$$T = 11\pi^p 0 \beta_1 \dots \beta_p \gamma_{W_1} \dots \gamma_{W_m}.$$

Each gadget β_j, γ_W is of length $\Theta(p)$, so $|T| = \Theta((p+m)^2)$. Moreover, T can clearly be constructed in $\Theta((p+m)^2)$ time. It suffices to prove that **W** is a YES-instance of the Universal Mismatch Problem if and only if $(T, 4p+3)$ is a YES-instance of the Shortest Cover in Partial Words Problem.

($\Rightarrow$) Suppose **W** is a YES-instance with a solution V. We shall prove that a solid string $S = 11h(V)0$ of length d is a cover of T. We have $S \approx 11\pi^p 0$ by definition of h and π; in particular S covers $11\pi^p 0$. Moreover, S covers each β_j by Corollary 2, and for each i it covers γ_{W_i} by Lemma 8 and due to the fact that $V \not\approx W_i$. Thus, T is a concatenation of partial words covered by S, and thus T itself is also covered by S.

($\Leftarrow$) Suppose that T has a solid cover S with $|S| \le d$. Clearly, $|S| > 1$ since both 0 and 1 occur as solid symbols in T. Thus, S begins with 11. Note that 11 does not occur in T at any position p with $1 < p \le d$. Consequently, S cannot be shorter than d, i.e., $S \approx 11\pi^p 0$.

By Observations 7 and 8, S occurs both as a prefix and as a suffix of each gadget words β_j and γ_W. It also covers their superstring T, so S covers each of the gadget words. By Corollary 2, $S = 11h(V)0$ for some partial word V, and by Lemma 8, V does not match any of the partial words $W_1, \dots, W_m$. □

Theorem 5. *The* Shortest Cover in Partial Words *Problem is NP-complete even for the binary alphabet.*

Proof. Equivalence between the CNF-SAT Problem and Universal Mismatch Problem (Observation 6) and the reduction above imply that the Shortest Cover in Partial Words Problem is NP-hard. It belongs to NP, since checking whether a given solid string is a cover can be implemented in polynomial time. □

The Exponential Time Hypothesis (ETH) [20,27] asserts that for some $\varepsilon > 0$ the 3-CNF-SAT Problem cannot be solved in $O(2^{\varepsilon p})$ time, where p is the number of variables. By the Sparsification Lemma [21,27], ETH implies that for some $\varepsilon > 0$ the 3-CNF-SAT Problem cannot be solved in $O(2^{\varepsilon(p+m)})$ time, and consequently in $2^{o(p+m)}$ time, where m is the number of clauses. Thus, Observation 6 and Lemma 9 also imply the following result.

Theorem 6. *Unless the Exponential Time Hypothesis is false, there is no $2^{o(\sqrt{n})}$-time algorithm for the* Shortest Cover in Partial Words *Problem. In particular, there is no $2^{o(\sqrt{k})}n^{O(1)}$-time algorithm for this problem.*

7. Conclusions

We considered the problems of finding the shortest solid cover of an indeterminate string and of a partial word. The main results of the paper are fixed-parameter tractable algorithms for these problems parameterized by k, that is, the number of non-solid symbols in the input. For the partial word covering problem we obtain an $O(nk^2 + 2^{O(\sqrt{k}\log k)})$-time algorithm whereas for covering a general indeterminate string we obtain an $O(nk^2 + 2^k k^3)$-time algorithm.

One open problem is whether the shortest cover of a general indeterminate string can be found as fast as the shortest cover of a partial word. Another question is to close the complexity gap for the latter problem, considering the lower bound resulting from the Exponential Time Hypothesis, which yields that no $2^{o(\sqrt{k})}n^{O(1)}$-time solution exists for this problem.

Acknowledgements

Tomasz Kociumaka is supported by Polish budget funds for science in 2013–2017 as a research project under the 'Diamond Grant' program. Jakub Radoszewski and Tomasz Waleń are supported by the Polish Ministry of Science and Higher Education under the 'Iuventus Plus' program in 2015–2017, grants no. 0392/IP3/2015/73 and 0179/DIA/2013/42. Jakub Radoszewski receives financial support of Foundation for Polish Science. Wojciech Rytter is supported by the Polish National Science Center, grant no 2014/13/B/ST6/00770.

References

[1] K.R. Abrahamson, Generalized string matching, SIAM J. Sci. Comput. 16 (6) (1987) 1039–1051, http://dx.doi.org/10.1137/0216067.

[2] A. Alatabbi, M.S. Rahman, W.F. Smyth, Computing covers using prefix tables, Discrete Appl. Math. 212 (2016) 2–9, http://dx.doi.org/10.1016/j.dam.2015.05.019.

[3] P. Antoniou, M. Crochemore, C.S. Iliopoulos, I. Jayasekera, G.M. Landau, Conservative string covering of indeterminate strings, in: J. Holub, J. Žďárek (Eds.), Prague Stringology Conference 2008, Czech Technical University, Prague, 2008, pp. 108–115, http://www.stringology.org/event/2008/p10.html.

[4] A. Apostolico, A. Ehrenfeucht, Efficient detection of quasiperiodicities in strings, Theoret. Comput. Sci. 119 (2) (1993) 247–265, http://dx.doi.org/10.1016/0304-3975(93)90159-Q.

[5] A. Apostolico, M. Farach, C.S. Iliopoulos, Optimal superprimitivity testing for strings, Inform. Process. Lett. 39 (1) (1991) 17–20, http://dx.doi.org/10.1016/0020-0190(91)90056-N.

[6] M.F. Bari, M.S. Rahman, R. Shahriyar, Finding all covers of an indeterminate string in o(n) time on average, in: J. Holub, J. Žďárek (Eds.), Prague Stringology Conference 2009, Czech Technical University, Prague, 2009, pp. 263–271, http://www.stringology.org/event/2009/p24.html.

[7] F. Blanchet-Sadri, Algorithmic Combinatorics on Partial Words. Discrete Mathematics and Its Applications, CRC Press, 2008, http://www.crcpress.com/product/isbn/9781420060928.

[8] D. Breslauer, An on-line string superprimitivity test, Inform. Process. Lett. 44 (6) (1992) 345–347, http://dx.doi.org/10.1016/0020-0190(92)90111-8.

[9] P. Clifford, R. Clifford, Simple deterministic wildcard matching, Inform. Process. Lett. 101 (2) (2007) 53–54, http://dx.doi.org/10.1016/j.ipl.2006.08.002.

[10] R. Cole, R. Hariharan, Verifying candidate matches in sparse and wildcard matching, in: J.H. Reif (Ed.), 34th Annual ACM Symposium on Theory of Computing, STOC 2002, ACM, 2002, pp. 592–601.

[11] M. Crochemore, C. Hancart, T. Lecroq, Algorithms on Strings, Cambridge University Press, New York, NY, USA, 2007.

[12] M. Crochemore, C.S. Iliopoulos, T. Kociumaka, J. Radoszewski, W. Rytter, T. Waleń, Covering problems for partial words and for indeterminate strings, in: H. Ahn, C. Shin (Eds.), Algorithms and Computation, ISAAC 2014, in: Lecture Notes in Comput. Sci., vol. 8889, Springer, 2014, pp. 220–232.

[13] M.J. Fischer, M.S. Paterson, String matching and other products, in: R.M. Karp (Ed.), Complexity of Computation, in: SIAM–AMS Proceedings, vol. 7, AMS, Providence, RI, 1974, pp. 113–125.

[14] Z. Galil, R. Giancarlo, Parallel string matching with k mismatches, Theoret. Comput. Sci. 51 (1987) 341–348, http://dx.doi.org/10.1016/0304-3975(87)90042-9.

[15] J. Holub, W.F. Smyth, Algorithms on indeterminate strings, in: Proceedings of 14th Australasian Workshop on Combinatorial Algorithms, 2003, pp. 36–45.

[16] J. Holub, W.F. Smyth, S. Wang, Fast pattern-matching on indeterminate strings, J. Discrete Algorithms 6 (1) (2008) 37–50, http://dx.doi.org/10.1016/j.jda.2006.10.003.

[17] C.S. Iliopoulos, M. Mohamed, L. Mouchard, K. Perdikuri, W.F. Smyth, A.K. Tsakalidis, String regularities with don't cares, Nordic J. Comput. 10 (1) (2003) 40–51.

[18] C.S. Iliopoulos, D.W.G. Moore, K. Park, Covering a string, Algorithmica 16 (3) (1996) 288–297, http://dx.doi.org/10.1007/BF01955677.

[19] C.S. Iliopoulos, J. Radoszewski, Truly subquadratic-time extension queries and periodicity detection in strings with uncertainties, in: R. Grossi, M. Lewenstein (Eds.), Combinatorial Pattern Matching, CPM 2016, in: LIPIcs, vol. 54, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016, pp. 8:1–8:12.

[20] R. Impagliazzo, R. Paturi, On the complexity of k-SAT, J. Comput. System Sci. 62 (2) (2001) 367–375, http://dx.doi.org/10.1006/jcss.2000.1727.

[21] R. Impagliazzo, R. Paturi, F. Zane, Which problems have strongly exponential complexity?, J. Comput. System Sci. 63 (4) (2001) 512–530, http://dx.doi.org/10.1006/jcss.2001.1774.

[22] P. Indyk, Faster algorithms for string matching problems: matching the convolution bound, in: 39th Annual Symposium on Foundations of Computer Science, FOCS 1998, IEEE Computer Society, 1998, pp. 166–173.

[23] A. Kalai, Efficient pattern-matching with don't cares, in: D. Eppstein (Ed.), 13th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2002, ACM/SIAM, 2002, pp. 655–656, http://dl.acm.org/citation.cfm?id=545381.545468.

[24] T. Kociumaka, M. Kubica, J. Radoszewski, W. Rytter, T. Waleń, A linear time algorithm for seeds computation, in: Y. Rabani (Ed.), 23rd Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 2012, SIAM, 2012, pp. 1095–1112.

[25] G.M. Landau, U. Vishkin, Efficient string matching with k mismatches, Theoret. Comput. Sci. 43 (1986) 239–249, http://dx.doi.org/10.1016/0304-3975(86)90178-7.

[26] Y. Li, W.F. Smyth, Computing the cover array in linear time, Algorithmica 32 (1) (2002) 95–106, http://dx.doi.org/10.1007/s00453-001-0062-2.

[27] D. Lokshtanov, D. Marx, S. Saurabh, Lower bounds based on the Exponential Time Hypothesis, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 105 (2011) 41–72, http://bulletin.eatcs.org/index.php/beatcs/article/view/92.

[28] D. Moore, W.F. Smyth, Computing the covers of a string in linear time, in: D.D. Sleator (Ed.), 5th Annual ACM–SIAM Symposium on Discrete Algorithms, SODA 1994, ACM/SIAM, 1994, pp. 511–515, http://dl.acm.org/citation.cfm?id=314464.

[29] S. Muthukrishnan, K. Palem, Non-standard stringology: algorithms and complexity, in: 26th Annual ACM Symposium on Theory of Computing, STOC 1994, ACM, New York, NY, USA, 1994, pp. 770–779.

[30] W.F. Smyth, S. Wang, An adaptive hybrid pattern-matching algorithm on indeterminate strings, Internat. J. Found. Comput. Sci. 20 (6) (2009) 985–1004, http://dx.doi.org/10.1142/S0129054109007005.

Please cite this article in press as: M. Crochemore et al., Covering problems for partial words and for indeterminate strings, Theoret. Comput. Sci. (2017), http://dx.doi.org/10.1016/j.tcs.2017.05.026