Covering Problems for Partial Words and for Indeterminate Strings

Maxime Crochemore[1,2], Costas S. Iliopoulos[1,3], Tomasz Kociumaka[4⋆], Jakub Radoszewski[4⋆⋆], Wojciech Rytter[4,5], and Tomasz Waleń[4]

[1] Dept. of Informatics, King's College London, London, UK
[maxime.crochemore,c.iliopoulos]@kcl.ac.uk
[2] Université Paris-Est, France
[3] Faculty of Engineering, Computing and Mathematics, University of Western Australia, Perth, Australia
[4] Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Warsaw, Poland
[kociumaka,jrad,rytter,walen]@mimuw.edu.pl
[5] Faculty of Mathematics and Computer Science, Copernicus University, Toruń, Poland

Abstract. We consider the problem of computing a solid cover of an indeterminate string. An indeterminate string may contain non-solid symbols, each of which specifies a subset of the alphabet that could be present at the corresponding position. We also consider covering partial words, which are a special case of indeterminate strings where each non-solid symbol is a don't care symbol. We prove that both indeterminate string covering problem and partial word covering problem are NP-complete for binary alphabet and show that both problems are fixed-parameter tractable with respect to k, the number of non-solid symbols. For the indeterminate string covering problem we obtain a $2^{\mathcal{O}(k \log k)} + nk^{\mathcal{O}(1)}$-time algorithm. For the partial word covering problem we obtain a $2^{\mathcal{O}(\sqrt{k} \log k)} + nk^{\mathcal{O}(1)}$-time algorithm. We prove that, unless the Exponential Time Hypothesis is false, no $2^{o(\sqrt{k})} n^{\mathcal{O}(1)}$-time solution exists for this problem, which shows that our algorithm for this case is close to optimal. We also present an algorithm for both problems which is feasible in practice.

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 subset of Σ. Such a *non-solid* symbol indicates that 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 $\diamondsuit$ (other popular notation is $*$).

⋆ Supported by Polish budget funds for science in 2013-2017 as a research project under the 'Diamond Grant' program.
⋆⋆ The author receives financial support of Foundation for Polish Science.

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[6] 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 [10].

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 [9]. Faster algorithms for this case were afterwards given in [21, 15, 16]. Pattern matching for general indeterminate strings, known as generalized string matching, was first considered by Abrahamson [1]. Since then numerous variants of pattern matching in indeterminate strings were considered. There were also practical approaches to the original problem; see [10, 22] for some recent examples. A survey on partial words, related mostly to their combinatorics, can be found in a book by Blanchet-Sadri [6].

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 [3]. A *cover* of a classical string s is a string that covers all positions of s with its occurrences. Covers in classical strings were already extensively studied. A linear-time algorithm finding the shortest cover of a string was given by Apostolico et al. [4] and later on improved into an on-line algorithm by Breslauer [7]. A linear-time algorithm computing all the covers of a string was proposed by Moore & Smyth [20]. Afterwards an on-line algorithm for the all-covers problem was given by Li & Smyth [18]. Other types of quasiperiodicities are seeds [12, 17] and numerous variants of covers and seeds, including approximate and partial covers and seeds.

The main problem considered here is as follows: Given an indeterminate string, find the length of its shortest solid cover; see Fig. 1. We can actually compute a shortest solid cover itself and all the lengths of solid covers, at no additional cost in the complexity. However, for simplicity we omit the description of such extensions in this version of the paper.

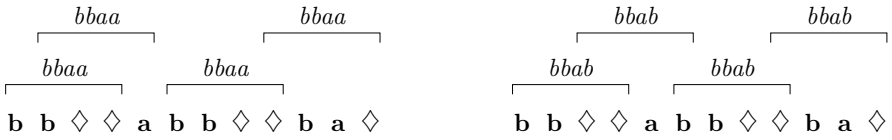

Fig. 1. Partial word $bb\diamondsuit\diamondsuit abb\diamondsuit\diamondsuit ba\diamondsuit$ with its two shortest covers. Note that the same non-solid symbol can match two different solid symbols for two different occurrences of the same cover.

Throughout the paper we use the following notations: n for the length of the given indeterminate string, k for the number of non-solid symbols in the input, and σ for the size of the alphabet. We assume that $2 \leq \sigma \leq n$ and that each

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

non-solid symbol in the indeterminate string is represented by a bit vector of size σ. Thus the size of the input is $\mathcal{O}(n + \sigma k)$.

The first attempts to the problem of indeterminate string covering were made in [2, 5, 11]. However, they considered indeterminate strings as covers and presented some partial results for this case. The common assumption of these papers is that $\sigma = \mathcal{O}(1)$; moreover, in [2, 5] the authors considered only so-called conservative indeterminate strings, for which $k = \mathcal{O}(1)$.

Our results: In Section 3 we show an $\mathcal{O}(n\sigma^{k/2}k^{\mathcal{O}(1)})$-time algorithm for covering indeterminate strings with a simple implementation. In Section 4 we obtain an $\mathcal{O}(2^{\mathcal{O}(k \log k)} + nk^{\mathcal{O}(1)})$-time algorithm. In the same section we devise a more efficient solution for partial words with $\mathcal{O}(2^{\mathcal{O}(\sqrt{k} \log k)} + nk^{\mathcal{O}(1)})$-time complexity. Finally in Section 5 we show that both problems are NP-complete already for binary alphabet. As a by-product we obtain that under the Exponential Time Hypothesis no $\mathcal{O}(2^{o(\sqrt{k})}n^{\mathcal{O}(1)})$-time solution exists for both problems.

2 Preliminaries

An *indeterminate string* (*i-string*, for short) T of length $|T| = n$ over a finite alphabet Σ is a sequence $T[1] \dots T[n]$ such that $T[i] \subseteq \Sigma$, $T[i] \neq \emptyset$. 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 i-string T and by σ we denote $|\Sigma|$. If $k = 0$, we call T a (solid) string. We say that two i-strings U and V *match* (denoted as $U \approx V$) if $|U| = |V|$ and for each $i = 1, \dots, |U|$ we have $U[i] \cap V[i] \neq \emptyset$.

Example 1. Let $A = a\,\{b, c\}$, $B = a\,\{a, b\}$, $C = aa$ be indeterminate strings (C is a solid string). Then $A \approx B$, $B \approx C$, however, $A \not\approx C$.

If all $T[i]$ 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 $\diamondsuit$.

By $T[i..j]$ we denote a factor $T[i] \dots T[j]$ of T. If $i = 1$ then it is called a prefix and if $j = n$ then it is called a suffix. We say that a pattern i-string S occurs in a text i-string T at position j if S matches $T[j..j + |S| - 1]$. A *border* of T is a solid string which matches both a prefix and a suffix of T. A border-length of T is a positive integer equal to the length of a border of T. A *cover* of T is a solid string S such that, for each $i = 1, \dots, n$, there exists an occurrence of S in T that contains the position i, i.e., an occurrence of S at one of the positions $\{i - |S| + 1, \dots, i\}$. Note that, just as in solid strings, every cover of T is also a border of T.

Example 2. The shortest cover of an i-string T need not be one of the shortest covers of the solid strings matching T. E.g., for the i-string $a\diamondsuit b$, where $\diamondsuit = \{a, b\}$, the shortest cover ab has length 2, whereas none of the solid strings aab, abb has a cover of length 2.

The following simple observation is an important tool in our algorithms.

Observation 3. *There are at most k^2 values (*shifts*) $i \in \{1, \ldots, n\}$ such that $T[1+\ell]$ and $T[i+\ell]$ are both non-solid for some ℓ.*

For convenience, we compute the set $T[i] \cap T[j]$ for each pair $T[i]$, $T[j]$ of non-solid symbols of T, and label different sets with different integers, so that afterwards we can refer to any of them in $\mathcal{O}(1)$ space. In particular, after such $\mathcal{O}(\sigma k^2)$-time preprocessing, we can check in $\mathcal{O}(1)$ time if any two positions of T match.

A longest common prefix (LCP) query in T, denoted as $\mathrm{lcp}(i, j)$, is a query for the longest matching prefix of the i-strings $T[i..n]$ and $T[j..n]$. Recall that for a solid string we can construct in $\mathcal{O}(n)$ time a data structure that answers LCP-queries in $\mathcal{O}(1)$ time, see [8]. Note that an LCP-query in an i-string can be reduced to $\mathcal{O}(k)$ LCP-queries in a solid string:

Lemma 4. *For an i-string with k non-solid symbols, after $\mathcal{O}(nk^2)$-time preprocessing, one can compute the LCP of any two positions in $\mathcal{O}(k)$ time.*

Lemma 4 lets us efficiently check if given pairs of factors of an i-string match and thus it has useful consequences.

Corollary 5. *Given i-strings S and T of total length n containing k non-solid symbols in total, one can compute $Occ(S, T)$, the list of all positions where S occurs in T, in $\mathcal{O}(nk^2)$ time.*

Corollary 6. *The set of all border-lengths of an i-string can be computed in $\mathcal{O}(nk^2)$ time.*

Note that a solid string of length at least $\frac{n}{2}$ is a cover of T if and only if it is a border of T. Therefore Corollary 6 enables us to easily solve the covering problem for cover lengths at least half of the word length. In the following sections we search only for the covers of length at most $\lfloor \frac{n}{2} \rfloor$.

3 Algorithm Parameterized by k and σ

Let T be an i-string of length n with k non-solid symbols. We assume that $T[1..\lfloor \frac{n}{2} \rfloor]$ contains at most $k/2$ holes (otherwise we reverse the i-string).

We say that S is a *solid prefix* of T if S is a solid string that matches $T[1..|S|]$. For an increasing list of integers $L = [i_1, i_2, i_3, \ldots, i_m]$, $m \geq 2$, we define

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

A set $\mathcal{P} \subseteq Occ(S, T)$ is a *covering set* for S if $\mathrm{maxgap}(\mathcal{P} \cup \{n+1\}) \leq |S|$, i.e., the occurrences of S at positions in $\mathcal{P}$ already cover the whole text T.

We introduce a $ShortestCover(S, L)$ subroutine which, for a given solid prefix S of T and an increasing list of positions L, checks if there is a cover of T which is a prefix of S for which the covering set is a sublist of L and, if so, returns the length of the shortest such cover. A pseudocode of this operation can be found on the next page. Correctness of the algorithm follows from the fact that

$$ShortestCover(S, L) = \min\Big\{j \ : \ \mathrm{maxgap}\Big(\bigcup_{t \geq j} L_t \cup \{n+1\}\Big) \leq j\Big\}.$$

Algorithm *ShortestCover*(S, L)
Input: S: a solid prefix of T; L: a sublist of $\{1, \ldots, n\}$
Output: The length of the shortest cover which is a prefix of S and has a covering set being a sublist of L
preprocessing:
foreach $i \in L$ **do** dist$[i]$:= lcp$(S, T[i..n])$;
$D := \{ \text{dist}[i] : i \in L \}$;
foreach $j \in D$ **do** $L_j := \{ i \in L : \text{dist}[i] = j \}$;
$L := L \cup \{n + 1\}$;
processing:
foreach $j \in D$ *in increasing order* **do**
if maxgap$(L) \leq j$ **then return** maxgap(L);
foreach $i \in L_j$ **do** remove i from L;
return *no solution;*

Lemma 7. *The algorithm ShortestCover(S, L) works in $\mathcal{O}(nk)$ time assuming that the data structure of Lemma 4 is accessible.*

Proof. Assume that each time we remove an element from the list we update maxgap(L). Then maxgap(L) may only increase. Each operation on the list L, including update of maxgap(L), is performed in $\mathcal{O}(1)$ time.

By Lemma 4, all lcp values can be computed in $\mathcal{O}(nk)$ time. The lists L_j can be computed easily in total time $\mathcal{O}(n)$. □

The shortest cover of T of length at most $\lfloor n/2 \rfloor$ is a prefix of a solid prefix of T of length $\lfloor n/2 \rfloor$. By the assumption made in the beginning of this section, T has at most $\sigma^{k/2}$ solid prefixes of length $\lfloor n/2 \rfloor$. For each of them we run the *ShortestCover*(S, L) algorithm with $L = \{1, \ldots, n\}$. Lemma 7 implies the following result.

Theorem 8. *The shortest cover of an i-string with k non-solid symbols can be computed in $\mathcal{O}(n\sigma^{k/2}k)$ time.*

4 Algorithm Parameterized by k

A border-length of T is called *ambiguous* if there are at least two different solid borders of T of this length, otherwise it is called unambiguous. A border of T is called unambiguous if it corresponds to an unambiguous border-length. By Observation 3, there are at most k^2 ambiguous border-lengths. The main idea of this section is to classify potential covers into two categories depending on whether the length is an unambiguous or an ambiguous border length.

Each unambiguous border is uniquely determined by its length. The solution for this case works in $\mathcal{O}(nk^4)$ time and uses the subroutine from Section 3. As for the second class, the number of ambiguous border-lengths is at most k^2. Hence, in this case the problem reduces to testing if there is a cover of a given length (this is still quite nontrivial; as we show later, the whole problem is NP-complete). In fact, the main difficulty is caused by the ambiguous borders.

Covering with Unambiguous Borders. For an i-string U of length m and a position i in T such that $U \approx T[i..i+m-1]$, we define:

$$U \odot i = U[1] \cap T[i], \ldots, U[m] \cap T[i+m-1].$$

Note that, for a prefix U of T, any i-string of the form $U \odot i$ can be represented in $\mathcal{O}(k)$ space (we only store the positions corresponding to non-solid symbols of U). Also every solid prefix of T has such a small representation. We call this a *sparse representation*.

Example 9. Let $T = bb\diamondsuit\diamondsuit abb\diamondsuit\diamondsuit baa$ and $U = b\diamondsuit a\diamondsuit$. Then

$U \odot 1 = U \odot 6 = bba\diamondsuit$, $U \odot 2 = b\diamondsuit aa$, $U \odot 7 = b\diamondsuit ab$, and $U \odot 9 = bbaa$.

The sparse representations of these i-strings are $(b, \diamondsuit)$, $(\diamondsuit, a)$, $(\diamondsuit, b)$ and (b, a).

A technical modification of the algorithm *ShortestCover* is required to show the following lemma. We omit its full proof in this version of the paper.

Lemma 10. *Let $\mathcal{C}$ be a collection of pairs (S, L), where each S is a solid prefix of T given in the sparse representation and each L is an increasing list of positions in T. If $|\mathcal{C}| \leq n$ and $\sum_{(S,L)\in\mathcal{C}} |L| = \mathcal{O}(nk^2)$ then ShortestCover(S, L) for all instances $(S, L) \in \mathcal{C}$ can be computed in $\mathcal{O}(nk^3)$ time.*

By $SolidOcc(U, T)$ ($NonSolidOcc(U, T)$) we denote the lists of all occurrences $i \in Occ(U, T)$ for which $U \odot i$ is a solid string (is not a solid string, respectively). All occurrences of U in T can be found using Corollary 5, and divided into these two sets in $\mathcal{O}(nk^2)$ time. By Observation 3, if U is a prefix of T then $|NonSolidOcc(U, T)| \leq k^2$.

Example 11. Let $U = a\diamondsuit$, $T = bb\diamondsuit\diamondsuit abb\diamondsuit\diamondsuit ba\diamondsuit$. Then

$$SolidOcc(U, T) = \{4, 5, 9\},\ NonSolidOcc(U, T) = \{3, 8, 11\}.$$

Theorem 12. *The shortest cover being an unambiguous border can be computed in $\mathcal{O}(nk^4)$ time.*

Proof. Let T be an i-string of length n and $p_1 < p_2 < \ldots < p_r$ be all non-solid symbols in its first half. Let $p_0 = 1$ and $p_{r+1} = \lfloor n/2 \rfloor + 1$. We divide all border-lengths into disjoint intervals $[p_j, p_{j+1} - 1]$, for $j = 0, \ldots, r$.

Consider the interval $I = [p_j, p_{j+1} - 1]$ and let $U = T[1..p_j]$. We compute the lists $E = SolidOcc(U, T)$ and $H = NonSolidOcc(U, T)$. Note that for each unambiguous border-length $d \in I$ we have $n - d + 1 \in E$.

We construct a set $\mathcal{C}$ of different pairs (S, L), where each S is of the form:

$$S = (U \odot i)\, T[p_j + 1\,..\,p_{j+1} - 1] \quad \text{for} \quad i \in E$$

and L is the list of occurrences of $U \odot i$ in E merged with the list H. If the shortest cover of T corresponds to an unambiguous border-length from I, it will be found in one of $ShortestCover(S, L)$ calls for $(S, L) \in \mathcal{C}$. Note that the lists L are disjoint on positions from E and $|H| \leq k^2$. We apply Lemma 10 for $\mathcal{C}$ to obtain $\mathcal{O}(nk^3)$ time for one instance I, U, and $\mathcal{O}(nk^4)$ time in total. □

Covering using Ambiguous Border-Lengths. In this section we are searching for a solid cover of T which matches its given prefix $U = T[1..m]$. We introduce the following auxiliary problem.

Problem 13. Given an i-string T, an integer m and a set $\mathcal{P} \subseteq Occ(T[1..m], T)$, find a solid cover $S \approx T[1..m]$ with a corresponding covering set $\mathcal{P}' \subseteq \mathcal{P}$ or state that no such S, $\mathcal{P}'$ exist.

In Lemma 14 we use the *ShortestCover* algorithm in a very similar way to the proof of Theorem 12. We omit the details.

Lemma 14. *Problem 13 can be solved in $\mathcal{O}(nk^3)$ time if the set $\mathcal{P}' \cap SolidOcc(T[1..m], T)$ is non-empty.*

For an integer $m \in \{1, \ldots, \lfloor n/2 \rfloor\}$ and a set of positions $\mathcal{P}'$, we introduce an auxiliary operation *TestCover*$(m, \mathcal{P}')$ which returns true iff there is a cover of T of length m for which $\mathcal{P}'$ is a covering set. This operation is particularly simple to implement for partial words; see the following lemma.

Lemma 15. *After $\mathcal{O}(2^{2k}k + nk^2)$-time preprocessing, TestCover$(m, \mathcal{P}')$ can be implemented in $\mathcal{O}(|\mathcal{P}'|k)$ time. If T is a partial word then $\mathcal{O}(nk^2)$-time preprocessing suffices.*

Proof. First consider the simpler case when T is a partial word. By definition, $\mathcal{P}'$ can be a covering set for a cover of length m if and only if $1 \in \mathcal{P}'$ and $\text{maxgap}(\mathcal{P}' \cup \{n+1\}) \leq m$. These conditions can be easily checked in $\mathcal{O}(|\mathcal{P}'|)$ 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 $i \in \mathcal{P}'$. After $\mathcal{O}(nk^2)$-time preprocessing, we can compute $\text{lcp}(1, i)$ for all $i \in \mathcal{P}'$ and check if each of those values is at least m. If not, then certainly such a string S does not exist. Otherwise, let the set Y contain positions of all don't care symbols in $T[1..m]$. We need to check, for each $j \in Y$, if the set

$$X_j = \{T[i-1+j] \,:\, i \in \mathcal{P}'\}$$

contains at most one solid symbol. This last step is performed in $\mathcal{O}(|\mathcal{P}'|\, k)$ time.

If T is a general i-string, the only required change is related to processing the X_j sets. If a set X_j contains a solid symbol, then it suffices to check if this symbol matches all the other symbols in this set. Otherwise we need some additional preprocessing.

Let Z be the set of all non-solid positions in T. We wish to compute, for each subset of Z, if there is a single solid symbol matching all the positions in this subset. For this, we first reduce the size of the alphabet. For each solid symbol $c \in \Sigma$, we find the subset of Z which contains this symbol. Note that if for two different solid symbols these subsets are equal, we can remove one of those symbols from the alphabet (just for the preprocessing phase). This way we reduce the alphabet size to at most 2^k. Afterwards we simply consider each subset of Z and look for a common solid symbol, which takes $\mathcal{O}(2^{2k}k)$ time. □

We use Lemma 15 to obtain a solution to Problem 13.

Lemma 16. *If $|\mathcal{P}| \leq k^2$, Problem 13 can be solved in $\mathcal{O}(2^{(2n/m)\log|\mathcal{P}|}nk/m + 2^{2k}k + nk^2)$ time or $\mathcal{O}(2^{(2n/m)\log|\mathcal{P}|}nk/m + nk^2)$ time if T is a partial word.*

Proof. Assume there is a solid string $S \approx T[1..m]$ for which there exists a covering set $\mathcal{P}' = \{i_1, \ldots, i_r\} \subseteq \mathcal{P}$ in T and further assume that $|\mathcal{P}'|$ is minimal. Notice that for each $j \in \{1, \ldots, r-2\}$, $i_{j+2} \geq i_j + m$. Indeed, otherwise $\mathcal{P}' \setminus \{i_{j+1}\}$ would also be a covering set for S in T. Hence, $r \leq 2n/m$.

In the algorithm we choose every subset $\mathcal{P}' \subseteq \mathcal{P}$ of size at most $\lfloor 2n/m \rfloor$ and run *TestCover*$(m, \mathcal{P}')$. By Lemma 15, the whole algorithm works in

$$\mathcal{O}\Big(2^{2k}k + nk^2 + \sum_{i \leq 2n/m} \tbinom{|\mathcal{P}|}{i} ik\Big) = \mathcal{O}\left(2^{2k}k + nk^2 + |\mathcal{P}|^{2n/m}\tfrac{n}{m}k\right)$$

$$= \mathcal{O}\left(2^{(2n/m)\log|\mathcal{P}|}nk/m + 2^{2k}k + nk^2\right)$$

time. If T is a partial word, the $2^{2k}k$ term can be dropped. □

Theorem 17. *The shortest cover of an i-string with k non-solid symbols can be computed in $\mathcal{O}(2^{\mathcal{O}(k\log k)} + nk^5)$ time.*

Proof. As candidates for the length of the shortest cover of a given i-string T of length n, we consider all border-lengths. By Theorem 12, we can consider all unambiguous border-lengths in $\mathcal{O}(nk^4)$ time. There are at most k^2 ambiguous border-lengths. If the cover of such a length m has an occurrence in $\mathit{SolidOcc}(T[1..m], T)$, due to Lemma 14 it can be computed in $\mathcal{O}(nk^3)$ time. Across all lengths this gives $\mathcal{O}(nk^5)$ time.

Note that, by the pigeonhole principle, T must contain a solid factor of length at least $\frac{n-k}{k+1}$. Thus, if $m \leq \frac{1}{2}\frac{n-k}{k+1}$, any cover of length m must have an occurrence within this factor, and consequently an occurrence in $\mathit{SolidOcc}(T[1..m], T)$. Therefore, if the cover has no occurrence in $\mathit{SolidOcc}(T[1..m], T)$, we have $m > \frac{n-k}{2k+2}$. If $k \leq \frac{n}{3}$ this concludes that $m > \frac{n}{3k+3}$ and consequently Lemma 16 for $\mathcal{P} = \mathit{NonSolidOcc}(T[1..m], T)$ yields an $\mathcal{O}(2^{\mathcal{O}(k\log k)} + nk^2)$-time algorithm. Otherwise $k = \Theta(n)$. Hence, Lemma 16 applied for $|\mathcal{P}| \leq k^2$ yields an $2^{\mathcal{O}(n\log k)} = 2^{\mathcal{O}(k\log k)}$-time solution. □

More Efficient Covering of Partial Words. The Exponential Time Hypothesis (ETH) [13, 19] asserts that for some $\varepsilon > 0$ the 3-CNF-SAT problem cannot be solved in $\mathcal{O}(2^{\varepsilon p})$ time, where p is the number of variables. By the Sparsification Lemma [14, 19], ETH implies that for some $\varepsilon > 0$ the 3-CNF-SAT problem cannot be solved in $\mathcal{O}(2^{\varepsilon(p+m)})$, and consequently in $2^{o(p+m)}$ time, where m is the number of clauses.

We show an algorithm for covering partial word which is more efficient than the generic algorithm for covering i-string. We also show that, unless ETH is false, our algorithm is not far from optimal.

Theorem 18.
(a) *The shortest cover of a partial word with k don't care symbols can be computed in $\mathcal{O}(2^{\mathcal{O}(\sqrt{k}\log k)} + nk^5)$ time.*
(b) *Unless the Exponential Time Hypothesis is false, there is no $2^{o(\sqrt{k})}n^{\mathcal{O}(1)}$-time algorithm computing the shortest cover of a partial word over binary alphabet.*

Proof. (a) We improve the algorithm from the proof of Theorem 17. The only part of that algorithm that does not work in $\mathcal{O}(nk^5)$ time is searching for a cover of length m being an ambiguous border-length of T, having all its occurrences in $H = \mathit{NonSolidOcc}(T[1..m], T)$. Recall that $|H| \leq k^2$. We solve this part more efficiently for partial word T.

Let $U = T[1..m]$. Let $\mathcal{P} \subseteq H$ be the set of positions such that $i \in \mathcal{P}$ if and only if $U \odot i$ has at most $\sqrt{k}$ don't care symbols. We consider two cases.

Case 1: the cover of length m has an occurrence $i \in \mathcal{P}$. Let $i_1, \ldots, i_r$ be the don't care positions in $U \odot i$. Let $M_1, \ldots, M_r$ be the sets of all solid symbols at positions $i_1, \ldots, i_r$ of $U \odot j$ for $j \in H$. If any of the sets M_a is empty, we insert an arbitrary symbol from Σ to it.

Let us construct all possible strings by inserting symbols from $M_1, \ldots, M_r$ at positions $i_1, \ldots, i_r$ in $U \odot i$. For each such solid string S, we simply compute a list L of all positions $j \in H$ such that $U \odot j \approx S$ and check if $1 \in L$ and if $\mathrm{maxgap}(L \cup \{n+1\}) \leq m$. Since $r \leq \sqrt{k}$ and $|M_a| \leq |H| \leq k^2$ for all $a = 1, \ldots, r$, this shows that Case 1 can be solved in $\mathcal{O}(k^{2\sqrt{k}+1}) = 2^{\mathcal{O}(\sqrt{k} \log k)}$ time.

Case 2: the cover of length m has all its occurrences in $H \setminus \mathcal{P}$. Let us divide T into $\lfloor \sqrt{k} \rfloor$ fragments of length at least $\lfloor n/\sqrt{k} \rfloor$ each. By the pigeonhole principle, at least one of those fragments contains at most $\lfloor \sqrt{k} \rfloor$ don't care symbols. No occurrence of the cover may be located totally inside this fragment. Therefore, $m \geq \lfloor \frac{n}{2\sqrt{k}} \rfloor$. Lemma 16 solves this case in $2^{\mathcal{O}(\sqrt{k} \log k)} + \mathcal{O}(nk^2)$ time.

(b) In Section 5 we show that the satisfiability problem (CNF-SAT) with p variables and m clauses can be reduced to finding the shortest cover of a binary partial word of length $n = \mathcal{O}((p+m)^2)$. Thus, unless ETH is false, the latter problem has no $2^{o(\sqrt{n})}$-time solution, i.e., no $2^{o(\sqrt{k})} n^{\mathcal{O}(1)}$ solution. □

5 Hardness of covering i-strings and partial words

The negative results obtained for partial words remain valid in the more general setting of the i-strings, so in this section we consider partial words only. We shall prove that the following decision problem is NP-complete.

Problem 19 (d-Cover in Partial Words*).* Given a partial word T of length n over an alphabet Σ and an integer d, decide whether there exists a solid cover S of T of length d.

We will reduce from the CNF-SAT problem. Recall that in this problem we are given a Boolean formula with p variables and 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 problem which is equivalent to the CNF-SAT problem, but more suitable for our proof.

*Problem 20 (*Universal Mismatch*).* Given 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 21. *The* Universal Mismatch *problem is equivalent to the CNF-SAT problem, and consequently it is NP-complete.*

Example 22. Consider the 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 $m = 3$ and five variables $(x_1, x_2, x_3, x_4, x_5)$. 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 $W_i[j] = \diamondsuit$ otherwise:

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

The interpretations $(1, 0, 1, 1, 0)$, $(1, 1, 1, 1, 0)$ satisfy ϕ. They correspond to partial words 10110, 11110 and $1\diamondsuit 110$, 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 $\mathcal{O}(p(p+m))$ which, as an instance of the d-Cover in Partial Words with $d = 4p + 2$, is equivalent to $\mathbf{W}$. Due to space constraints, we do not give a rigorous correctness proof of our construction.

We define a morphism

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

and construct T so that V is a solution to $\mathbf{W}$ if and only if $S = 11\,h(V)$ covers T. The word T is of the form $11\pi^p\beta_1 \ldots \beta_p\gamma_{W_1} \ldots \gamma_{W_m}$, where $\pi = 0\diamondsuit 0\diamondsuit$ and β_i, γ_W are *gadgets* to be specified later. These gadgets are chosen so that any d-cover of T must be a d-cover of each gadget string.

The prefix $11\pi^p$ of T enforces that any d-cover S of T is of the form $S = 11s_1 \ldots s_p$ where $s_j \approx \pi$ for each j. Thus, in order to make sure that S is of the form $h(V)$ for some partial word V, for each j we need to rule out the possibility that $s_j = 0101$. To this end, we define $\beta_j = 11\,\pi^{p-1}\,0\,\diamondsuit^{4j+1}\,00\,\diamondsuit^d$.

Lemma 23. *Let $S = 11s_1 \ldots s_p$ be a solid string with $s_i \approx \pi$ for each i. Then S covers β_j if and only if $s_j \neq 0101$.*

Consequently, the d-covers of $11\pi^p\beta_1 \ldots \beta_p$ are precisely the strings of the form $11h(V)$ for binary partial words V of length p.

We encode the constraints $V \not\approx W$ using gadgets $\gamma_W = 11\mu(W^R)01\diamondsuit^d$, where W^R denotes the reverse of a partial word W and μ is the following morphism:

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

Lemma 24. *Let V and W be binary partial words of length p. Then $11h(V)$ covers γ_W if and only if $V \not\approx W$.*

Note that $11h(V)$ covers γ_W if and only if it occurs in $\mu(W^R)01\diamondsuit^{4p}$. Thus the key idea behind the proof of Lemma 24 is the following *relation between μ and h*; see also Fig. 2.

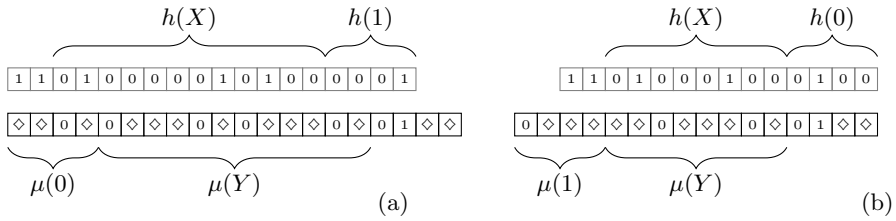

Fig. 2. Illustration of Lemma 25: an occurrence of $11h(Xc)$ in $\mu(c'Y)\,01\diamondsuit\diamondsuit$ for (a) $Xc = 0101$, $c'Y = 01\diamondsuit0$; (b) $Xc = 000$, $c'Y = 100$. In general, $11h(Xc)$ is a prefix of $\mu(c'Y)\,01\diamondsuit\diamondsuit$ if $c = 1$ and $c' = 0$, and a suffix — if $c = 0$ and $c' = 1$.

Lemma 25. *Let $c, c' \in \{0, 1, \diamondsuit\}$, and let X, Y be partial words of the same length. Then $11h(Xc)$ occurs in $\mu(c'Y)\,01\diamondsuit\diamondsuit$ if and only if $c \not\approx c'$.*

The reduction described above shows that the d-Cover in Partial Words problem, restricted to the binary alphabet, is NP-hard. Clearly, this problem also belongs to NP, which yields the main result of this section.

Theorem 26. *The d-Cover in Partial Words problem is NP-complete even for the binary alphabet.*

6 Conclusions

We considered the problems of finding the length of 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 $\mathcal{O}(2^{\mathcal{O}(\sqrt{k}\log k)} + nk^{\mathcal{O}(1)})$-time algorithm whereas for covering a general indeterminate string we obtain an $\mathcal{O}(2^{\mathcal{O}(k\log k)} + nk^{\mathcal{O}(1)})$-time algorithm. The latter can actually be improved to $\mathcal{O}(2^{\mathcal{O}(k)} + nk^{\mathcal{O}(1)})$ time by extending the tools used in the proof of Theorem 18. In all our algorithms a shortest cover itself and all the lengths of covers could be computed without increasing the complexity.

One open problem is to determine if the shortest cover of indeterminate strings can be found as fast as the shortest cover of partial words. 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 $2^{o(\sqrt{k})}n^{\mathcal{O}(1)}$-time solution exists for this problem.

References

1. Abrahamson, K.R.: Generalized string matching. SIAM Journal on Computing 16(6), 1039–1051 (1987)

2. Antoniou, P., Crochemore, M., Iliopoulos, C.S., Jayasekera, I., Landau, G.M.: Conservative string covering of indeterminate strings. In: Holub, J., Žďárek, J. (eds.) Prague Stringology Conference 2008. pp. 108–115. Czech Technical University, Prague (2008)
3. Apostolico, A., Ehrenfeucht, A.: Efficient detection of quasiperiodicities in strings. Theoretical Computer Science 119(2), 247–265 (1993)
4. Apostolico, A., Farach, M., Iliopoulos, C.S.: Optimal superprimitivity testing for strings. Information Processessing Letters 39(1), 17–20 (1991)
5. Bari, M.F., Rahman, M.S., Shahriyar, R.: Finding all covers of an indeterminate string in $O(n)$ time on average. In: Holub, J., Žďárek, J. (eds.) Prague Stringology Conference 2009. pp. 263–271. Czech Technical University, Prague (2009)
6. Blanchet-Sadri, F.: Algorithmic Combinatorics on Partial Words. Chapman & Hall/CRC Press, Boca Raton, FL (2008)
7. Breslauer, D.: An on-line string superprimitivity test. Information Processing Letters 44(6), 345–347 (1992)
8. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press (2007)
9. Fischer, M.J., Paterson, M.S.: String matching and other products. In: Karp, R.M. (ed.) Complexity of Computation. SIAM-AMS Proceedings, vol. 7, pp. 113–125. AMS, Providence, RI (1974)
10. Holub, J., Smyth, W.F., Wang, S.: Fast pattern-matching on indeterminate strings. Journal of Discrete Algorithms 6(1), 37–50 (2008)
11. Iliopoulos, C.S., Mohamed, M., Mouchard, L., Perdikuri, K., Smyth, W.F., Tsakalidis, A.K.: String regularities with don't cares. Nordic Journal of Computing 10(1), 40–51 (2003)
12. Iliopoulos, C.S., Moore, D., Park, K.: Covering a string. Algorithmica 16(3), 288–297 (1996)
13. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. Journal of Computer and System Sciences 62(2), 367–375 (2001)
14. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? Journal of Computer and System Sciences 63(4), 512–530 (2001)
15. Indyk, P.: Faster algorithms for string matching problems: Matching the convolution bound. In: 39th Annual Symposium on Foundations of Computer Science. pp. 166–173. IEEE Computer Society, Los Alamitos, CA (1998)
16. Kalai, A.: Efficient pattern-matching with don't cares. In: Eppstein, D. (ed.) 13th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 655–656. SIAM, Philadelpha, PA (2002)
17. Kociumaka, T., Kubica, M., Radoszewski, J., Rytter, W., Waleń, T.: A linear time algorithm for seeds computation. In: Rabani, Y. (ed.) 23rd Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 1095–1112. SIAM, Philadelpha, PA (2012)
18. Li, Y., Smyth, W.F.: Computing the cover array in linear time. Algorithmica 32(1), 95–106 (2002)
19. Lokshtanov, D., Marx, D., Saurabh, S.: Lower bounds based on the Exponential Time Hypothesis. Bulletin of the EATCS 105, 41–72 (2011)
20. Moore, D., Smyth, W.F.: Computing the covers of a string in linear time. In: Sleator, D.D. (ed.) 5th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 511–515. SIAM, Philadelpha, PA (1994)
21. Muthukrishnan, S., Palem, K.V.: Non-standard stringology: algorithms and complexity. In: 26th Annual ACM Symposium on Theory of Computing. pp. 770–779. ACM, New York, NY (1994)
22. Smyth, W.F., Wang, S.: An adaptive hybrid pattern-matching algorithm on indeterminate strings. International Journal of Foundations of Computer Science 20(6), 985–1004 (2009)