The Hardness of Counting Full Words Compatible with Partial Words

Florin Manea\textsuperscript{a,b,*}, Cătălin Tiseanu\textsuperscript{c}

\textsuperscript{a}Christian-Albrechts-Universität zu Kiel, Institut für Informatik, D-24098 Kiel, Germany.
\textsuperscript{b}University of Bucharest, Faculty of Mathematics and Computer Science, RO-010014 Bucharest, Romania.
\textsuperscript{c}University of Maryland at College Park, Computer Science Department, A.V. Williams Bldg., College Park, MD 20742, USA

Abstract

We present several problems regarding counting full words compatible with a set of partial words or with the factors of a partial word, and show that they are \#P-complete. Some of these counting problems have NP-complete decision counterparts to which a hard variant of CNF-SAT is reduced parsimoniously; the rest are \#P-complete problems that cannot be canonically associated to NP-complete decision problems. For these problems we assume that the set of symbols compatible with the wildcards equals the alphabet of the input partial word. When both a partial word and the cardinality of the alphabet compatible with the wildcard are given as input, we show that the central problem of counting the full words compatible with factors of the given partial word is also \#P-complete. Finally, we propose a nontrivial exponential-time algorithm, working in polynomial space, useful to derive upper bounds for the time needed to solve the discussed problems.

An extended abstract of this paper was presented at the conference Language and Automata, Theory and Applications 2010, [1].

\*Corresponding Author.

Email addresses: flm@informatik.uni-kiel.de (Florin Manea), ctiseanu@umd.edu (Cătălin Tiseanu)

Preprint
of this word, and counting all the distinct $k$-repetitions compatible with at least a factor of a given partial word. To get all these results we made the natural assumption that the set of symbols that can replace the hole is equal to, or strictly contained in, the alphabet of the input partial word. In the case when we get as input not only a partial word, but also the cardinality of the set of symbols that can replace the hole, we show that counting all the distinct full words compatible with factors of the given partial word is also #P-complete. Finally, as a counterpart to the lower bounds shown in the paper, we propose a nontrivial exponential-time algorithm, working in polynomial space, that solves the problem of counting the number of words compatible with at least one word from a given list of partial words; this algorithm can be used to derive upper bounds for the time needed to solve most of the discussed problems.

**Keywords:** Partial Words, NP-completeness, #P Complexity Class, #P-complete Problems, Combinatorics on Words.

**2000 MSC:** 68Q17, 68W40, 68W32, 68Q45

1. Introduction

Partial words are sequences that, besides regular symbols, may have a number of unknown symbols, called holes or wildcards, generalizing, in this way, the classical notion of words. The study of the combinatorial properties of partial words began with the paper [2], of Berstel and Boasson, and it was motivated by an intriguing practical problem, gene comparison, which relates to some central topics of combinatorics on words. Until now, several such combinatorial properties of partial words have been investigated: periodicity, conjugacy, freeness and primitivity (see [3] for an extensive survey and further references on such works). Part of these studies consisted in finding efficient algorithms testing if a word and its factors verify certain combinatorial properties ([4, 5, 6, 7, 8]).

An appealing research direction in the study of partial words, related to those mentioned already, consists in identifying and counting specific factors of partial words: for instance, identifying and counting the distinct repetitions in a partial word, or the primitive factors of a partial word, etc.. However, in the case of counting problems ([9, 10]) one is usually interested in finding the number of all the different full words, satisfying a specific condition, that are compatible with factors of a given partial word (for instance,
square full words that are compatible with factors of a given partial word, or full words of a fixed length that are compatible with factors of a given partial word, etc.) instead of counting the actual factors (partial words, at their turn) of the given partial word. This approach seems natural, as one can easily imagine a scenario where two or more distinct factors of a partial word are pairwise compatible, so counting its actual factors would not give an exact image on the expressiveness of that partial word (assuming that a word that has more distinct factors is seen as more expressive than one with fewer); instead, identifying or counting the full words compatible with the factors of a given partial word seems to provide a better image on how “expressive” that partial word really is.

Until now, most of the results obtained in this area state mathematical properties of functions that express the result of such counting problems. Here, we are interested in the computational aspects of several such counting problems. More precisely, we want to devise lower and upper bounds on the time needed to solve these problems computationally. To this end, we prove that several problems where one is interested in counting the full words that are compatible with the partial words from a list or where one is interested in counting full words that are compatible with factors of a partial word (that may be required to verify specific properties) are complete for the class \#P. Thus, these problems are computationally hard. Also, we propose an exponential-time algorithm, working in polynomial space, that can be used to solve all the discussed problems, in a relatively efficient manner. It is worth stressing the fact that in most cases identifying and counting distinct partial words that appear as factors (possibly with specific properties) of a partial word is computationally easy (see [6, 8]); however, as soon as we are interested in counting the full words compatible with such factors the corresponding problems become computationally hard.

The structure of our paper is the following. In the second section we give some basic definitions and preliminary facts. In the third section we show that a series of problems on partial words are NP-complete, and we derive hardness results for the counting problems that can be canonically associated with them. One of these results, which is worth noting, is that computing the subword complexity both for finite and infinite partial words is a computationally hard problem. Next we propose several counting problems, that cannot be associated canonically with NP-complete problems, but are still complete for the class \#P. We end the part of the paper that deals with lower bounds by showing that another problem (counting all the distinct full
words that are compatible with the factors of a word) is also \#P-complete, but in a more general setting where we are given as input both the input word and the size of the alphabet of symbols that can replace the holes (in the rest of the paper, this alphabet is considered to be exactly the set of symbols that appear in the input word). The paper ends with a section in which we describe a nontrivial exponential-time algorithm, working in polynomial-space, that solves one of the problems presented in this paper, and briefly explain how it can be used to solve the other problems.

2. Basic definitions

Let us first recall some basic denotations. An alphabet $V$ is a finite set of symbols. Any finite sequence of symbols from an alphabet $V$ is called full word (or, simpler, word) over $V$. By $V^*$ we denote the set of all full words (strings) over $V$ (including the empty word $\lambda$). The length of a full word $w$ is denoted by $|w|$, while $|w|_a$ denotes the number of occurrences of the symbol $a \in V$ in the word $w$. By $V^+$ and $V^L$ for some natural number $L$ we denote the set of all non-empty full words and the set of all full words with length $L$, respectively.

A partial word of length $n$ over the alphabet $V$ is a partial function $u : \{1, \ldots, n\} \rightarrow V$. For $i \in \{1, \ldots, n\}$, if $u(i)$ is defined (hence $u(i) \in V$) we say that $i$ belongs to the domain of $u$ (denoted by $i \in D(u)$), otherwise we say that $i$ belongs to the set of holes of $u$ (denoted by $i \in H(u)$).

Let $\diamond$ be a symbol that does not belong to $V$. For convenience, finite partial words are seen as full words over the extended alphabet $V \cup \{\diamond\}$ (see [3]); a partial word $u$ of length $n$ is depicted as $u = a_1 \cdots a_n$, where $a_i = u(i)$, for $i \in D(u)$, and $a_i = \diamond$, otherwise. A partial word over $V$ whose set of holes is empty can be seen as a full word from $V^*$. In this way, one can easily define the concatenation of partial words, as the concatenation of the corresponding full words over $V \cup \{\diamond\}$, and the length of partial words, as the length of the corresponding full words over $V \cup \{\diamond\}$; all the other notions defined for full words can be similarly extended for the case of partial words. We denote by $\lambda$ the empty partial word (i.e., the partial word of length 0).

The partial words $u$ and $v$ are said to be equal if $u$ and $v$ have the same length, $D(u) = D(v)$ and $u(i) = v(i)$ for all $i \in D(u)$. If $u$ and $v$ are two partial words of equal length, then $u$ is said to be contained in $v$, $u \subseteq v$, if all the elements of $D(u)$ are contained in $D(v)$ and $u(i) = v(i)$ for all $i \in D(u)$. 

4
Note that, for a full word \( u \) and a partial word \( v \) with \( |u| = |v| \), if \( u \subseteq v \) then \( H(v) = \emptyset \) and \( u = v \).

Similarly to the classical case of full words (see, for instance, [11]), we say that the partial word \( u \) is a factor of the partial word \( w \) if there exist partial words \( x \) and \( y \) such that \( w = xuy \). If \( x = \lambda \) we say that \( u \) is a prefix of \( w \), and if \( y = \lambda \) we say that \( u \) is a suffix of \( w \). If \( w = a_1 \cdots a_n \), we denote by \( w[i..j] \) the factor \( a_i \cdots a_j \) of \( w \), and by \( w[i] \) the symbol \( a_i \); we say that \( w[i] \) is the symbol placed on the \( i^{th} \) position in the partial word \( w \).

We say that two partial words \( u \) and \( v \) are compatible, denoted by \( u \uparrow v \), if the two words agree in all positions where they are both defined, i.e., there exists a full word \( w \) such that \( u \subseteq w \) and \( v \subseteq w \).

Let \( w \in (V \cup \{\diamond\})^* \) be a partial word; \( w \) is said to be a \( k \)-repetition if \( w = x_1 \cdots x_k \) and there exists a non-empty partial word \( u \) such that \( x_i \subseteq u \) for all \( i \in \{1, \ldots, k\} \). In the case of full words, a \( k \)-repetition over \( V \) is a word having the form \( x^k \) with \( x \in V^* \). Usually, 2-repetitions are called squares; in the case of full words, a square over \( V \) is a word having the form \( xx \) with \( x \in V^+ \).

The reader interested in more definitions and results on partial words is referred to the handbook [3].

For the definitions regarding the computational complexity notions appearing in this paper, such as different complexity classes, NP-complete problems, polynomial-time reductions and Turing reductions, we refer to the classical handbook [12], or to the more recent [13, 14]. For the definition and for some seminal results regarding the complexity class \#P and \#P-complete problems we refer to [15]; we just briefly recall that \#P is the class of function problems that ask to “compute \( f(x) \)”, for the input \( x \), provided that \( f \) is the number of accepting paths of a non-deterministic polynomial Turing machine. Note that NP-completeness is meant with respect to polynomial-time many-one reductions, while \#P-completeness is meant with respect to Turing reductions.

We also recall the definition of the basic NP-complete problem CNF-SAT (Satisfiability of Boolean Formulas in conjunctive normal form).

**Problem 1.** Given \( f \) a Boolean formula in conjunctive normal form, with the variables \( S = \{x_1, \ldots, x_k\} \), i.e., \( f = C_1 \land C_2 \land \cdots \land C_n \) where each \( C_i \) is the disjunction of literals (variables from \( S \) or the negation of these variables), decide whether there exists an assignment of the variables from \( S \) that makes \( f \) true.
The natural counting problem associated with CNF-SAT, usually denoted by \(#CNF\text{-SAT}\), asks how many assignments of the variables from \(S\), that make \(f\) true, exist. This problem is \(#P\)-complete.

In this paper, a natural variant of CNF-SAT is used. A formula is said to be in restricted conjunctive normal form (denoted in the following CNF\(^*\)) if it does not contain any clause in which both a variable and its negation are present. Clearly, the problem of deciding the satisfiability of a CNF\(^*\) formula (denoted CNF\(^*\)-SAT in this paper) is NP-complete; a polynomial-time many-one reduction from CNF-SAT to this problem is immediate. Indeed, the satisfiability of a CNF formula is equivalent to the satisfiability of the CNF\(^*\) formula obtained by deleting, from the original formula, all the clauses in which a variable and its negation appear (which are clearly satisfied). The counting problem associated with CNF\(^*\)-SAT, denoted \(#CNF^*\text{-SAT}\), is also \(#P\)-complete. A polynomial-time Turing reduction from \(#CNF\)-SAT to \(#CNF^*\)-SAT is also immediate. We first do the aforementioned deletion that transforms the initial CNF formula into a CNF\(^*\) formula, remember the number \(n\) of variables that appeared in the original formula but do not appear in the CNF\(^*\) formula anymore, and obtain that the number of satisfying assignments for the initial CNF formula equals the number of satisfying assignments for the CNF\(^*\) formula times \(2^n\) (as the variables that do not appear anymore can be assigned any value).

3. A series of NP-complete problems and the associated counting problems

First, let us consider the following basic problem.

**Problem 2.** Given a list of partial words \(S = \{w_1, w_2, \ldots, w_k\}\) over the alphabet \(V\) with \(|V| \geq 2\), each partial word having the same length \(L\), decide whether there exists a word \(v \in V^L\) such that \(v\) is not compatible with any of the partial words in \(S\).

We show that this problem is NP-complete by showing that the CNF\(^*\)-SAT problem can be reduced to it in polynomial time.

**Theorem 1.** Problem 2 is NP-complete.

**Proof.** First, assume that we are given a list \(S = \{w_1, w_2, \ldots, w_k\}\) of partial words over the alphabet \(V\), each partial word having the same length \(L\).
We can easily construct a non-deterministic Turing machine $M$, working in polynomial time, that decides whether there exists a full word $v \in V^L$ such that $v$ is not compatible with any of the partial words in $S$. The machine $M$ non-deterministically constructs a full word $v \in V^L$; this can be done in linear time. Then, it checks (deterministically, this time) whether the full word $v$ is compatible with any of the partial words in $S$ or not; again, this takes linear time. If the check reveals that the constructed word $v$ is compatible with one of the input words the machine rejects, otherwise it accepts. Clearly, the non-deterministic machine described above works in linear time on any input. Thus, Problem 2 is in NP. In the following we show that the problem is also complete for the class NP.

Let us consider an instance of the CNF*-SAT Problem. More precisely, let $f$ be a Boolean formula in restricted conjunctive normal form, let $L$ be the number of logical variables which appear in $f$ and denote these variables by $x_1, x_2, \ldots, x_L$, let $n$ be the number of clauses of $f$ and let $C_1, C_2, \ldots, C_n$ be these clauses (each of them being actually the disjunction of several literals). Then, if $f = C_1 \land C_2 \land C_3 \land \cdots \land C_n$, it is immediate that an assignment of the variables $x_1, \ldots, x_L$ makes $f$ equal to 1 if and only if the same assignment of the variables $x_1, \ldots, x_L$ makes $\bar{f}$ (the negation of $f$) equal to 0. Note that $\bar{f} = \bar{C}_1 \lor \bar{C}_2 \lor \cdots \lor \bar{C}_n$. We can now associate the following instance of Problem 2 with the instance of the CNF*-SAT problem defined by the boolean formula $f$. Consider the alphabet $V = \{0, 1\}$, the length of the words $L$, and construct the list of partial words $S = \{w_1, w_2, \ldots, w_n\}$, where $w_i$, with $i \in \{1, \ldots, n\}$, is defined as follows:

$$
\text{for } j \in \{1, \ldots, L\}, \text{ let } w_i[j] = \begin{cases} 
0, & \text{if } \overline{x}_j \in \overline{C}_i; \\
1, & \text{if } x_j \in \overline{C}_i; \\
\diamond, & \text{otherwise.}
\end{cases}
$$

It is clear that a word of length $L$ over $V$, denoted by $v$, corresponds to an assignment of the variables $\{x_1, \ldots, x_L\}$, and conversely, in a canonical way: we simply take $x_j = v[j]$, for all $j \in \{1, \ldots, L\}$. One can show that, for a given $i$, an assignment of the variables $\{x_1, \ldots, x_L\}$ makes $\overline{C}_i = 1$ if and only if the word that corresponds to that assignment is compatible with $w_i$. Indeed, if a word $v \in V^L$ is compatible with a partial word from the list $S$, say $w_i$, then all the literals that appear in $\overline{C}_i$ are equal to 1, thus, the variables assignment defined by $v$ makes $\overline{C}_i$ equal to 1. Conversely, an assignment of the variables that makes $\overline{C}_i$ equal to 1 (i.e., makes all the literals of $\overline{C}_i$ equal to 1) corresponds to a word, denoted by $v$, which is compatible with
It follows immediately that for a given $i$, an assignment of the variables \( \{x_1, \ldots, x_L\} \) makes \( \overline{C_i} = 0 \) if and only if the word corresponding to that assignment is not compatible with \( w_i \). This shows that deciding whether there exists an assignment of the variables that makes \( \overline{f} \) equal to 0 (consequently, making \( f \) equal to 1) corresponds to deciding whether there exists a full word which is not compatible with any of the partial words of the set \( S \), thus, it corresponds to solving Problem 2 for the list \( S \).

Finally, notice that the reduction described above (from an instance of CNF*-SAT to an instance of Problem 2) can be easily implemented by a deterministic Turing machine working in polynomial time. Consequently, it follows that Problem 2 is NP-complete. Furthermore, this reduction clearly establishes a bijection between the solutions of the initial instance of CNF*-SAT and the solutions of the instance of Problem 2 we construct; therefore, this reduction is parsimonious. This final remark becomes useful in the proof of Theorem 2.

Now consider the counting problem associated with Problem 2:

**Problem 3.** Given a list of partial words \( S = \{w_1, w_2, \ldots, w_k\} \) over the alphabet \( V \), with \( |V| \geq 2 \), each partial word having the same length \( L \), count the distinct words \( v \in V^L \) such that \( v \) is compatible with at least one of the partial words in \( S \).

From Theorem 1 we can derive that this problem is \#P-complete as follows.

**Theorem 2.** Problem 3 is \#P-complete.

**Proof.** One can easily modify the non-deterministic Turing machine described in the beginning of the proof of Theorem 1 such that it has, for a given input, as many accepting paths as the number of words in \( V^L \) that are compatible with at least one of the partial words of the input list. Indeed, such a machine constructs non-deterministically a word from \( V^L \) and then checks (deterministically) whether it is compatible with at least one of the partial words in \( S \) or not. If it is compatible with such a word the machine accepts the input, otherwise it rejects the input. Once again, this machine works in linear time, and the possible computation paths correspond bijectively to the full words constructed non-deterministically at the beginning of the computation (i.e., the words of \( V^L \)); moreover, an accepting computation path corresponds to a full word of \( V^L \) that is compatible with at least one of the partial words in \( S \). Therefore, Problem 3 is in \#P.
The proof of Theorem 1 shows that \#CNF*-SAT can be Turing-reduced in polynomial time to the following problem: given a list of partial words \( S = \{w_1, w_2, \ldots, w_k\} \) over the alphabet \( V \), with \( |V| \geq 2 \), each partial word having the same length \( L \) count the distinct words \( v \in V^L \) such that there exists no partial word in \( S \) compatible with \( v \). It follows that this problem is \#P-complete. Moreover, this problem can be also Turing-reduced in polynomial time, by a canonical reduction, to Problem 3. More precisely, the result of Problem 3 is obtained by subtracting from \( |V|^L \) the number of distinct words from \( V^L \) that are not compatible with any of the partial words in \( S \). Consequently, this problem is also \#P-complete. \( \square \)

In the following we address a problem regarding the factors of a given length of a partial word.

**Problem 4.** Given a partial word \( w \) over the alphabet \( V \), with \( |V| \geq 2 \), and a natural number \( L \) with \( 0 < L \leq |w| \) decide whether there exists a word \( v \in V^L \) such that \( v \) is not compatible with any factor of length \( L \) of \( w \).

We show that this problem is NP-complete, by reducing Problem 2 to it.

**Theorem 3.** Problem 4 is NP-complete.

**Proof.** By arguments similar to the ones used in the previous proofs, Problem 4 is in NP. It remains to show that it is also complete for this class.

In this respect we consider an instance of Problem 2. Let \( S = \{w_1, w_2, \ldots, w_k\} \) be a list of partial words over the alphabet \( V \) each of them having the same length \( L \). As \( |V| \geq 2 \), we can choose \( a \) and \( b \), two distinct symbols from \( V \). We consider the partial word

\[ w = abw_1ba^{L+1}bw_2ba^{L+1}\cdots bw_kba^{L+1}a. \]

Clearly, this word can be constructed from the aforementioned instance of Problem 2 by a deterministic Turing machine working in polynomial time. We show that there exists a word \( v \in V^L \) which is not compatible with any of the partial words in \( S \) if and only if there exists a word \( v' \in V^{L+2} \) such that \( v' \) is not compatible with any factor of length \( L + 2 \) of \( w \). Once we prove this statement it follows that we have a deterministic polynomial-time many-one reduction from Problem 2 to Problem 4, and, since Problem 2 is NP-complete, it follows that Problem 4 is also NP-complete.

In order to finish the proof, let us analyse which words from \( V^{L+2} \) are compatible with factors of \( w \). First of all, it is clear that all the full words of
length \( L + 2 \) that start or end with \( a \) are compatible with a factor of \( w \), for instance with the factor \( a \diamond L+1 \) or with the factor \( \diamond L+1 a \), respectively. Then, we notice that every word of length \( L + 2 \) over \( V \) that neither starts nor ends with \( a \) and is compatible with a factor of \( w \) must be compatible with at least one of the partial words \( b w_1 b, b w_2 b, \ldots, b w_k b \); thus, if there exists a word \( v \) from \( V^L \) which is not compatible with any of the partial words from \( S \), then \( b v b \) is not compatible with any factor of \( w \), and conversely. This concludes our proof.

Again, we can consider the counting problem associated with Problem 4:

**Problem 5.** Given a partial word \( w \) over the alphabet \( V \), with \( |V| \geq 2 \), and a natural number \( L \), with \( 0 < L \leq |w| \), count the distinct words \( v \in V^L \) such that \( v \) is compatible with at least a factor of length \( L \) of \( w \).

As in the former case, it follows that this problem is \#P-complete.

**Theorem 4.** Problem 5 is \#P-complete.

**Proof.** One can easily show that Problem 5 is in \#P. The polynomial-time many-one reduction from the previous proof can be easily used to produce a polynomial-time Turing reduction from Problem 3 to Problem 5.

Assume that \( \text{Count}(w, L) \) is the counting function defined in Problem 5, i.e., \( \text{Count}(w, L) \) is the number of distinct words \( v \in V^L \) such that \( v \) is compatible with a factor of length \( L \) of \( w \). Now, consider an instance of Problem 3. Let \( S = \{ w_1, w_2, \ldots, w_k \} \) be a list of partial words over the alphabet \( V \) each of them having the same length \( L \), and assume that we want to compute the number of words from \( V^L \) which are compatible with at least one word of \( S \), denoted \( X_{S,L} \). As in the previous proof, consider the partial word

\[
w = a b w_1 b a^{L+1} b w_2 b a^{L+1} \cdots b w_k b a^{L+1} \diamond^{L+1} a.
\]

We have shown that a word \( v \in V^L \) is compatible with at least one of the partial words in \( S \) if and only if the full word \( b v b \in V^{L+2} \) is compatible with at least a factor of length \( L + 2 \) of \( w \), and all the other full words of length \( L + 2 \) which are compatible with at least a factor of \( w \) are the words from the set \( a V^{L+1} \cup V^{L+1} a \). Therefore, \( X_{S,L} = \text{Count}(w, L) - (2 |V|^{L+1} - |V|^L) \). This shows that Problem 3 can be Turing-reduced in polynomial time to Problem 5. Therefore, Problem 5 is also \#P-complete. \( \square \)

Theorems 3 and 4 have many implications. First, one can show the following result, using similar reductions:
Corollary 1. Consider the following problems:

(i). Given a partial word $w$ over the alphabet $V$, with $|V| \geq 2$, and a natural number $L$ with $0 < L \leq |w|$ decide whether there exists a natural number $\ell$, with $0 < \ell \leq L$, and a word $v \in V^\ell$ such that there exists no factor of $w$ compatible with $v$.

(ii). Given a partial word $w$ over the alphabet $V$ with $|V| \geq 2$ and a natural number $L$, with $0 < L \leq |w|$, count the full words $v \in V^L$, with $0 < \ell \leq L$, such that there exists no factor of $w$ compatible with $v$.

Problem (i) is NP-complete and Problem (ii) is $\#P$-complete.

Proof. One can use the same reductions as in the proofs of Theorem 3 and Theorem 4 to show (i) and (ii), respectively. We do not go into details once more. However, the key remark showing that these reductions are indeed useful is that the partial word

$$w = abw_1ba^{L+1}bw_2ba^{L+1}\ldots bw_kb^{L+1}a,$$

constructed in the aforementioned proofs, has the factor $\cdot^{L+1}$, which is compatible with any full word strictly shorter than $L + 2$. Thus, counting the full words $v \in V^{L+2}$ such that there exists no factor of $w$ compatible with $v$ (or deciding the existence of such a full word) can be reduced to counting the full words $v \in V^\ell$, with $0 < \ell \leq L + 2$, such that there exists no factor of $w$ compatible with $v$ (or deciding the existence of such a word, respectively).

Both Problem 5 and Problem (ii) stated in Corollary 1 are related to the subword complexity of a word, as defined in [11]. The subword complexity of a full word is defined for finite and right infinite words as follows: let $V$ be a finite alphabet and $w$ be a finite or right infinite word over $V$; the subword complexity of $w$ is the function which assigns to each positive integer $n$ the number $p_w(n)$ of distinct factors of length $n$ of $w$. One can give a similar definition for partial words ([10]): let $V$ be a finite alphabet and $w$ be a finite or right infinite partial word over $V$; the subword complexity of $w$ is the function which assigns to each positive integer $n$ the number $p_w(n)$ of distinct full words over $V$ that are compatible with at least a factor of length $n$ of $w$.

A direct consequence of Theorem 4 is that computing the subword complexity of a finite partial word is $\#P$-hard. More precisely, the problem
“given a partial word $x$ compute $p_x(k)$, for all $k \leq |x|$”, which can be seen as a function problem (i.e., a problem in which we are interested in computing the value of a function when its argument is the input), and which is trivially contained in FP$\#P$ (the class of function problems solved in deterministic polynomial time by Turing machines that have access to a $\#P$ oracle), is $\#P$-hard.

Also, we can consider a class of very simple right infinite partial words: let $V$ be an alphabet, with $|V| \geq 2$, and let $C = \{w\diamond L^{-1} \diamond L^{-1} \cdots | w \in (V \cup \{\diamond\})^*, \$ \notin V, 0 < L \leq |w|\}$. Clearly, one can compute the value $p_x(n)$ for every word $x \in C$ and every natural number $n \in \mathbb{N}$. Moreover, each word $x \in C, x = w\diamond L^{-1} \diamond L^{-1} \cdots$ for some $w \in (V \cup \{\diamond\})^*$, can be described succinctly in the following manner: we consider the morphism $\phi : V \cup \{$, $\diamond$} $\rightarrow (V \cup \{$, $\diamond$})*, defined by $\phi(a) = a$, for all $a \in V$, $\phi(\diamond) = \diamond$ and $\phi(\$) = $\diamond L^{-1}\$; clearly $x = \lim_{n \rightarrow \infty} \phi^n(\$)$ and the space needed to represent $x$ in this way is $O(|w|)$. Now we can consider the problem of computing the subword complexity of the infinite partial words from the class $C$: “given $x \in C$ and $n \in \mathbb{N}$, compute $p_x(k)$, for all $k \leq n$”. However, it is not hard to see that if $n \geq L$ solving this restricted problem implies solving Problem 5 for the partial word $w$ and the number $L$. As we have already shown, Problem 5 is $\#P$-complete, thus, the problem “given $x \in C$ and $n \in \mathbb{N}$, $n \geq L$, compute $p_x(k)$, for all $k \leq n$” is $\#P$-hard. Consequently, computing the subword complexity of the infinite partial words from the class $C$ is $\#P$-hard. Further, this shows that computing the subword complexity of an infinite partial word (when it is possible) is a hard counting problem.

Finally, one may be interested in counting all the full words that are compatible with at least a factor of a partial word $w$ over an alphabet with at least 2 symbols. We were not able to show neither that this problem can be solved efficiently nor that it is a hard counting problem. However, we conjecture that it is a $\#P$-complete problem, as well. In this respect, the result of Theorem 5 shows that a natural approach would not yield an efficient solution of the problem: one cannot hope to solve it efficiently by counting separately the factors of length $k$ of the partial word $w$, for all $k \leq |w|$, and summing up the results afterwards. Also, the results shown in Section 5 show that a natural generalization of this problem is $\#P$-complete, as well.
4. Other hard counting problems for partial words

Let us note that if a counting problem asks to count the partial words that verify a certain property we can canonically associate with this problem two decision problems, in which we have to decide whether there exists a word, or, respectively, whether there is no word, that verifies the given property. The hard counting problems that we presented in the last section have all a common feature: they can be associated canonically with (and were actually derived from) hard decision problems. But, as stated in [15], the most interesting hard counting problems are those that can not be associated canonically with a hard decision problem.

Consider, for instance, the open problem mentioned in the end of the previous section: count all the full words that are compatible with factors of a partial word \( w \) over an alphabet with at least 2 symbols. It is clear that one can efficiently decide whether there exists or not a full word, of length less or equal to \( n \), that is not compatible with any factor of the partial word \( w \), where \( w \in (V \cup \{\diamond\})^n \). If \( w \) does contain a symbol \( a \), other than \( \diamond \), at position \( i \), we construct a word of the same length with \( w \) having the symbol \( b \) at position \( i \), where \( b \in V \setminus \{a\} \) and this word is not compatible with any factor of \( w \); otherwise, when \( w \) contains only \( \diamond \) symbols, any full word of length less or equal to \( n \) is compatible with a factor of \( w \). Also, if \( n > 0 \) then there exists always a word compatible with a factor of \( w \): if \( w \) contains a symbol \( a \) different from \( \diamond \), then \( a \) is such a word; otherwise, any word of length 1 is compatible with a factor of \( w \). Thus, the decision problems associated canonically with the counting problem we mentioned are not hard.

In the following we present a series of other counting problems that cannot be associated, in the manner described above, with hard decision problems, but which can be shown to be \#P-complete. While the first three are related somehow to the problem of counting all the distinct full words compatible with the factors of a partial word, the last two come from the area of combinatorics on words, being related to the problem of counting distinct squares (or, more general, repetitions) in a partial word (see [9]).

The first problem we approach is strongly related to Problem 5 and Problem (ii) from Corollary 1. We are interested in counting, for a partial word \( w \) and a natural number \( L \) with \( 0 < L \leq |w| \), the number of distinct full words that are compatible with factors of \( w \), of length at least \( L \).

**Problem 6.** Given a partial word \( w \) over the alphabet \( V \), with \( |V| \geq 2 \), and a natural number \( L \), \( 0 < L \leq |w| \), count the full words \( v \in V^* \) with
that are compatible with at least a factor of \( w \).

A similar reasoning as the above shows that this problem cannot be associated canonically with an NP-complete problem. However, it is \( \#P \)-complete.

**Theorem 5.** Problem 6 is \( \#P \)-complete.

**Proof.** It is rather plain to see that this problem is in \( \#P \). Let a partial word \( w \) over the alphabet \( V \) and a natural number \( L \) with \( L \leq |w| \) be an instance of our problem. A non-deterministic polynomial Turing machine that accepts the full words \( v \in V^* \), with \( L \leq |v| \leq |w| \), such that \( v \) is compatible with a factor of \( w \) non-deterministically chooses a word from \( V^* \), shorter than \( w \) but with length greater than \( L \), and checks (deterministically) if it is compatible with a factor of \( w \); if so the input is accepted, otherwise it is rejected. Such a machine clearly has as many accepting paths as the number of solutions of Problem 6 for the given input. Consequently, this problem is in \( \#P \). It remains to show that it is complete for this class.

Assume that Solve\((w, L)\) is the counting function defined in Problem 6, that is, Solve\((w, L)\) equals the number of words \( v \in V^* \) with \( L \leq |v| \leq |w| \) that are compatible with a factor of \( w \). Also, consider the partial word \( w \) and the natural number \( L \) as an input instance of Problem 5. It is not hard to see that a solution of Problem 5 is to return the value Solve\((w, L)\)-Solve\((w, L + 1)\) (making the convention that Solve\((w, |w| + 1)\) returns 0). This shows that Problem 5, which is \( \#P \)-complete according to Theorem 4, can be Turing-reduced in polynomial time to Problem 6. In conclusion, this problem is also \( \#P \)-complete.

The second problem consists in counting all the full words over a restricted alphabet that are compatible with the factors of a partial word.

**Problem 7.** Given a partial word \( w \) over the alphabet \( V \), with \( |V| \geq 3 \), and a symbol \( \$ \in V \) count the full words \( v \in (V \setminus \{\$\})^* \), with \( 0 < |v| \leq |w| \), that are compatible with at least a factor of \( w \).

This problem cannot be associated canonically with an NP-complete problem. The existence of a word over \( V \setminus \{\$\} \) compatible with a factor of \( w \) is easy to settle: if \( w \) contains a \( \diamond \) symbol or a symbol \( a \in V \setminus \{\$\} \) then \( a \) is such a word; otherwise, if \( w \) contains only \( \$ \) symbols, then such a word does not exist. The existence of a word over \( V \setminus \{\$\} \) which is not compatible with any factor of \( w \) is again easy: if \( w \) does contain a symbol \( a \in V \) at position \( i \) then we construct a word of the same length with \( w \) having the symbol \( b \).
at position $i$, where $b \in V \setminus \{a, \$$\}$, and this word is not compatible with any factor of $w$; otherwise, i.e., $w$ contains only $\diamond$ symbols, any full word over $V \setminus \{\$$\}$ of length less or equal to $|w|$ is compatible with a factor of $w$.

However, we show that Problem 7 is a hard counting problem, by giving a Turing reduction from Problem 3.

**Theorem 6.** Problem 7 is $\#P$-complete.

**Proof.** It is not hard to see that this problem is in $\#P$. We show that it is also complete for this class.

Let $\text{Solve}(w, \$$)$ be the counting function defined in Problem 7, that is, $\text{Solve}(w, \$$)$ equals the number of words $v \in (V \setminus \{\$$\})^*$, with $0 < |v| \leq |w|$, that are compatible with a factor of $w$. Also, consider an instance of Problem 3: $S = \{w_1, w_2, \ldots, w_k\}$ is a list of partial words of length $L$ over the alphabet $V \setminus \{\$$\}$, which has at least two symbols. We construct the partial word

$$w = w_1\$$w_2\$$\cdots\$$w_k\diamond^{L-1}\$$L-1\$.$$

It is not hard to see that $\text{Solve}(w, \$$)$ has the value $\sum_{1 \leq \ell \leq L-1} (|V| - 1)^\ell + N_L$, where $N_L$ is the number of words over $V \setminus \{\$$\}$ compatible with at least a word from the list $S$. Indeed, any full word over $V$ of length $\ell$, where $\ell < L$, is compatible with a factor $\diamond^\ell$, and the full words over $V$ of length $L$ compatible with a factor of $w$ are exactly the full words over $V$ which are compatible with one of $w_1$, $w_2$, $\ldots$, $w_k$. Thus, $N_L$ can be obtained by subtracting $\sum_{1 \leq \ell \leq L-1} (|V| - 1)^\ell$ from the value of $\text{Solve}(w, \$$)$. Consequently, we have shown that Problem 3 can be Turing-reduced, in polynomial time, to Problem 7. Since Problem 3 is $\#P$-complete, it follows that Problem 7 is also $\#P$-complete.

In the following, we consider a restricted compatibility relation. Given two partial words $u$ and $v$ over the alphabet $V$, $|V| \geq 3$, and a symbol $s \in V$, we say that $u$ and $v$ are compatible-$s$ (read as: the words are "compatible minus $s$"), denoted by $u \uparrow_s v$, if there exists a partial word $w$ such that $u \subseteq w$ and $v \subseteq w$ and for each $i \in H(u) \cup H(v)$ we have $w[i] \neq s$. Intuitively, the idea behind this compatibility relation is that the $\diamond$ symbol is seen as a wildcard that can be replaced by any symbol of the alphabet different from $s$; in the usual case $\diamond$ can be replaced by all the symbols of the alphabet.

Now we consider the problem of counting, for a partial word $w$ over $V$ and a symbol $\$$ \in V$, all the full words that are compatible-$\$$ with a factor of $w$. 

15
Problem 8. Given a partial word \( w \) over the alphabet \( V \), with \( |V| \geq 3 \), and a symbol \( \$ \in V \) count the full words \( v \in V^* \), with \( 0 < |v| \leq |w| \), that are compatible-$ with at least a factor of \( w \).

One can show, similarly to the case of Problem 7, that Problem 8 cannot be associated canonically with an NP-complete problem: it can be efficiently decided whether there exists a word \( v \in V^* \), with \( 0 < |v| \leq |w| \), compatible-$ with a factor of \( w \), or whether there exists a word shorter than \( w \) which is not compatible-$ with any factor of \( w \). Further, we show that Problem 8 is also \#P-complete.

Theorem 7. Problem 8 is \#P-complete.

Proof. Again, it is not hard to see that this problem is in \#P. It remains to show that it is complete for \#P.

Assume that the function \( \text{Comp}(w, \$) \) is the counting function defined in Problem 8, i.e., \( \text{Comp}(w, \$) \) equals the number of words \( v \in V^* \), with \( 0 < |v| \leq |w| \), that are compatible-$ with a factor of \( w \). Further, consider an input instance of Problem 5: \( w \) is a partial word over the alphabet \( V \setminus \{\$\} \) (note that \( |V \setminus \{\$\}| \geq 2 \), and \( L \) is a natural number with \( 0 < L \leq |w| \); we should compute the number \( X_{w,L} \) of the full words over \( V \setminus \{\$\} \) compatible with at least a factor of length \( L \) of \( w \).

Consider now the partial words \( w_1 = \diamond^{L-1}w \) and \( w_2 = \diamond^Lw \). We briefly analyse the full words that are compatible-$ with factors of these two words:

- All the full words from \( (V \setminus \{\$\})^\ell \), with \( 0 < \ell \leq L - 1 \), are compatible-$ with factors of both \( w_1 \) and \( w_2 \), namely with factors of the form \( \diamond^\ell \).

- A full word from \( (V \setminus \{\$\})^L \) is compatible-$ with a factor of \( w_1 \) if and only if it is compatible with a factor of \( w \). On the other hand, all the full words from \( (V \setminus \{\$\})^L \) are compatible-$ with a factor of \( w_2 \), namely with \( \diamond^L \). Thus, exactly \( X_{w,L} \) full words over \( V \setminus \{\$\} \) are compatible-$ with factors of length \( L \) of \( w_1 \) and \((|V| - 1)^L \) full words over \( V \setminus \{\$\} \) are compatible-$ with factors of \( w_2 \).

- A full word from \( (V \setminus \{\$\})^\ell \), with \( L < \ell \leq L + |w| \), is compatible-$ with a factor of \( w_1 \) if and only if it is compatible with a factor of \( w \) (note that such a word has the length less than or equal to \( |w| \)). The same holds for the full words from \( (V \setminus \{\$\})^\ell \), with \( L < \ell \leq L + |w| + 1 \),
compatible-§ with factors of \( w_2 \). Therefore, the number of the full words from \((V \setminus \{\$\})^\ell\), with \( L < \ell \leq L + |w| \), which are compatible-§ with a factor of \( w_1 \) equals the number of the from \((V \setminus \{\$\})^\ell\), with \( L < \ell \leq L + |w| + 1 \), compatible-§ with a factor of \( w_2 \).

- A full word of length \( \ell \) over \( V \), with \( 1 \leq \ell \leq L + |w| \), containing §, is compatible-§ with a factor of \( w_1 \) if and only if it is compatible-§ with a factor of \( w_2 \). These words have the form \( x\$y \), with \( x \in V^p \), for some \( p < \ell \), and \( y \in V^{\ell-p-1} \) compatible with a prefix \( v' \) of \( w \); they are compatible with \( \diamond^p$v'$.

- There are no full words of length \( L + |w| + 1 \) compatible-§ with factors of \( w_1 \), but there are several such words compatible-§ with the entire \( w_2 \). The number of these words is denoted by \( N_w \) and equals \((|V| - 1)^L\), if \( w \) contains no \( \diamond \) symbol, or \((|V| - 1)^L + |w|\), otherwise (since the \( \diamond \)-symbols cannot be replaced by §). \( N_w \) can be computed in polynomial time, starting from \( w \).

From the above considerations it follows that:
\[
\text{Comp}(w_2, \$) - \text{Comp}(w_1, \$) = (|V| - 1)^L - X_{w,L} + N_w.
\]

Therefore, we have:
\[
X_{w,L} = (|V| - 1)^L - (\text{Comp}(w_2, \$) - \text{Comp}(w_1, \$)) + N_w.
\]

Since the number \( N_w \) can be computed in polynomial time we have obtained a polynomial-time Turing reduction from Problem 5 to Problem 8. Since Problem 5 is \#P-complete, it follows that Problem 8 is \#P-complete, as well.

We continue by proving that counting all the square full words which are compatible with factors of a partial word is also a hard counting problem.

**Problem 9.** Given a partial word \( w \) over the alphabet \( V \), with \(|V| \geq 2\), count the full words \( x \in V^* \), with \( 0 < |x| \leq |w| \) and \( x = vv \) for some \( v \in V^* \), compatible with at least a factor of \( w \).

According to the results in [6] one can identify all the 2-repetitions in a partial word \( w \) in \( \mathcal{O}(|w|^2) \) computational time. Thus, one can decide efficiently the existence of a square \( vv \) which is compatible with at least one factor of \( w \). On the other hand, for a word \( w \) of even length there exists also a square
that is not compatible with any of its factors unless \( w \) has only \( \odot \) symbols; for a word \( w \) of odd length there exists also a square that is not compatible with any of its factors unless \( w \) has the form \( \odot^{2k}a \) or \( a\odot^{2k} \) for some positive number \( k \) and \( a \in V \cup \{\odot\} \) (the arguments are similar with those used in the case of Problems 7 and 8). Therefore, Problem 9 cannot be associated canonically with an \( \text{NP} \)-complete problem. However, this problem is also hard for the class \#P.

**Theorem 8.** Problem 9 is \#P-complete.

**Proof.** It is straightforward to construct a non-deterministic polynomial Turing machine having as many accepting paths as the number of words \( x \in V^* \), with \( 0 < |x| \leq |w| \) and \( x = vv \) for some \( v \in V^* \), compatible with a factor of \( w \). Thus, Problem 9 is in \#P. It remains to show that it is also complete for this class.

We finish this proof by giving a reduction from a slightly modified version of Problem 3. Let the function \( \text{Squares}(w) \) be the counting function defined in Problem 9, i.e., \( \text{Squares}(w) \) equals the number of words \( x \in V^* \), with \( 0 < |x| \leq |w| \) and \( x = vv \) for some \( v \in V^* \), compatible with a factor of \( w \). Further, consider an input instance of Problem 3: \( S = \{w_1, w_2, \ldots, w_k\} \), with \( k \geq 3 \), is a list of partial words of length \( L \) over the alphabet \( V = \{0, 1\} \); we are interested in computing all the full words over \( V' = V \cup \{\$, \$, \$, \$, \$, \$\} \), which are compatible with at least one word of the list (this version of Problem 3 can be shown to be \#P-complete in the exact same way as in the case of the initial problem).

Starting from the partial words of the list \( S \) we can construct, in deterministic polynomial time, the partial word \( w \), given by:

\[
w = \odot^{4L+1}w_1\odot^{kL+1}\odot^{4L+2}w_2\odot^{kL+2}\odot^{4L+3}w_3\odot^{kL+3}\odot^{4L+4}w_4\ldots \odot^{4L+k}w_k\odot^{kL+k} \odot^{2L+2} \odot^{2L+2}.
\]

Next we analyse what square full words can be compatible with factors of \( w \):

**a.** All the words \( x = vv \) from \( V'^{2\ell} \), with \( \ell \leq L + 1 \), are compatible at least with a factor of the form \( \odot^{2\ell} \) of \( w \). The number of these words is \( N_1 = \sum_{k=1}^{L+1} 7^k \). Also, all the words of the form \( \odot^{2\ell} \) with \( L+1 < \ell \leq \lfloor k/2 \rfloor \) and \( \odot^{2t} \) with \( L+1 < t \leq \lfloor (kL+k)/2 \rfloor \) are squares contained in \( w \); their number is \( N'_1 = \max\{0, \lfloor k/2 \rfloor -(L+1)\}+\max\{0, \lfloor (kL+k)/2 \rfloor -(L+1)\}. \)
b. If $v$ is a word of length $L$, compatible with one of the partial words $w_1, w_2, \ldots, w_k$, then \( v^*_L \) is a square compatible with a factor of $w$. The number of such squares equals the number of words over $V'$ compatible with at least a word from the list $S$, denoted here by $X_{S,V'}$.

c. All the words of the form $\dagger^2r$, with $k^2L^2 + 1 > r > L + 1$, are squares compatible with factors of $w$. Their number is, clearly, $N_2 = k^2L^2$.

d. All the words of the form $vv$, where $v = \dagger^r x$, $L + 1 < r + |x|$, $r + 2|x| \leq 2L + 2$, $r > 0$, $|x| > 0$, and $x$ starts with a symbol different from $\dagger$, are squares compatible with factors of $w$ (more precisely with factors of the form $\dagger^r \circ 2|x+r|$). If we denote by $t$ the length of $v$ and by $r_t = \max\{1, 2t - (2L + 2)\}$, the number of such words is given by the relation: $N_3 = \sum_{L+1 < t < 2L+2} \sum_{r_t < r < t} (6 \cdot 7^{t-r-1})$.

e. Any other word of length greater than $2L + 2$ contained in $w$ is not a square (we show this a little later). Moreover, the sets of squares described in the previous four claims are pairwise disjoint.

Clearly, the numbers $N_1, N'_1, N_2,$ and $N_3$ can be computed in polynomial time. Also, $X_{S,V'} = \text{Square}(w) - N'_1 - N_1 - N_2 - N_3$. But this shows a polynomial-time Turing reduction from Problem 3 or Problem 9. Therefore, Problem 9 is $\#P$-complete.

It remains only to show that the first part of the claim $e$ above is true. To show this claim, it is sufficient to consider the case when $w_j = \circ^k$ for all $j \in \{1, \ldots, k\}$; clearly, if one of the partial words $w_j$ contains symbols that differ from $\circ$ then the partial word $w$ contains fewer squares, of length greater than $2L + 2$, than in the case when all its symbols are equal to $\circ$. For this, let $vv$ be a square, compatible with a factor of $w$, other than any of the squares mentioned in the claims a,b,c,d. There exists a factor $x_1x_2$ of $w$ such that $x_1 \subseteq v$ and $x_2 \subseteq v$. Let $y$ be the starting symbol of $x_1$ and $z$ be the starting symbol of $x_2$. There are several cases to be analysed:

1. $y = \S$ and $z = \S$. If $x_1 = \S w_1 \dagger \S w_1 \dagger \cdots \S \perp^{j-1}$ it follows that $x_2 = \S w_2 \dagger \S w_2 \dagger \cdots$; but this is impossible due to the fact that $x_2$ contains more $S$ symbols, after the second $\dagger$, than $x_1$ contains after the second $\dagger$, thus, one of the $S$ symbols in $x_2$ should be compatible with a $\perp$ symbol from $x_1$, a contradiction. If $x_1 = \S w_1 \dagger \S w_1 \dagger \cdots \S \perp^{j-1} w_2 \dagger$ it follows that $x_2 = \S w_2 \dagger \S \cdots$; again, this is impossible due to the fact
that the second $\triangledown$ symbol of $x_1$ is incompatible with a $\$ symbol from $x_2$. If $x_1 = \triangledown w_1 \triangledown \triangledown \ldots \triangledown w_j \triangledown w_j \triangledown \cdot \cdot \cdot$ it follows that $x_2 = \triangledown w_j \triangledown \cdot \cdot \cdot$; this is a contradiction, again, because a $\$ symbol of $x_2$ is incompatible with a $\perp$ symbol of $x_1$. Finally, if $x_1 = \triangledown w_1 \triangledown \perp \triangledown w_j \cdot \cdot \cdot \perp j \cdot \cdot \cdot$ it follows that $x_2 = \triangledown w_j \triangledown \perp \cdot \cdot \cdot \perp \cdot \cdot \cdot$; this is a contradiction, because the first $\$ symbol from $x_1$ is incompatible with a $\triangledown$ symbol from $x_2$. Clearly, no other case exists.

2. $y = \triangledown$ and $z = \circ$. If $x_1 = \triangledown w_1 \triangledown \triangledown \cdot \cdot \cdot \perp^j \triangledown \cdot \cdot \cdot$ it follows that $x_2 = \cdot \cdot \cdot \perp^j \cdot \cdot \cdot$, where $\perp u = w_j$; this leads to a contradiction, since the first $\perp$ symbol in $x_1$ is in compatible with the $\$ symbols from $x_2$. If $x_1 = \triangledown w_1 \triangledown \triangledown \cdot \cdot \cdot \perp^j \triangledown \cdot \cdot \cdot$ it follows that $x_2 = \cdot \cdot \cdot \perp^j \cdot \cdot \cdot$, where $\perp u = w_j$; in this case, the first $\perp$ from $x_2$ is incompatible with a $\$ symbol from $x_1$, thus, a contradiction. Finally, $x_1$ cannot contain a $\dagger$ since it would imply that $x_1$ contains all the $\dagger$ symbols, and, thus, it would be longer than $x_2$. This completes the analysis of this case.

3. The cases when $y \in \{\dagger, \perp, \$\}$ and $z = y$ or $z = \circ$ can be treated similarly to the above. In the case when $y = \$$, the factor $x_1 x_2$ can only be a full word of the form $\$^{2\ell}$ with $L + 1 < \ell \leq kL + k$, but these words were already taken into account in claim a; there are no other factors $x_1 x_2$ that are compatible with a square and start with $\$$. Similarly, if $y = \perp$, the factor $x_1 x_2$ can only be a full word of the form $\perp^{2\ell}$ with $L + 1 < \ell \leq k$, and these words were also taken into account in claim a; no other factors $x_1 x_2$ that are compatible with a square and start with $\perp$ exist. When $y = \dagger$ there are no factors $x_1 x_2$ that fulfill the above conditions.

The key idea in showing these facts is (as it can be seen in the cases described in details, above) that we always reach a situation when two different symbols from $\{\dagger, \triangledown, \perp, \circ, \$\}$ should be compatible, thus, a contradiction. This situation occurs because the number of $\$ and $\perp$ symbols that follow each block $\triangledown w_i \triangledown \cdot \cdot \cdot \triangledown w_j$ is different for different values of $i$, making impossible the matching of a symbol $a \in \{\dagger, \triangledown, \perp, \circ, \$\}$ with the exact same symbol of $x_2$; thus, $a$ should be matched with a $\circ$, but this is also impossible, since it leads to a situation where a $\$ symbol from one of $x_1$ or $x_2$ is matched with a different symbol from $\{\dagger, \triangledown, \perp, \circ, \$\}$ from $x_2$ or $x_1$, respectively. Also, $y$ cannot be $\dagger$, since we would get that $vv$ is one of the full words.
analysed in the claims a,b,c,d.

4. \( y = \diamond \) and \( z = \lozenge \).

- If \( x_1 = \diamond u_1 \frac{\lozenge}{j} w_i \cdots \frac{\perp}{j-1} u_2 \) it follows that \( x_2 = \diamond u_3 \frac{\lozenge}{j} w_j \cdots \), where there exists \( u_0 \) such that \( u_0 \diamond u_1 = w_i \) and \( u_2 \diamond u_3 = w_j \). If \( |u_1| = |u_3| \) we reach a contradiction because the first \( \perp \) symbol of \( x_1 \) would be compatible with a \( \lozenge \) symbol of \( x_2 \). If \( |u_1| < |u_3| \) the second \( \frac{\lozenge}{j} \) symbol of \( x_2 \) would be compatible with a \( \lozenge \) symbol from \( x_1 \), again a contradiction.

- If \( x_1 = \diamond u_1 \frac{\lozenge}{j} w_i \cdots \frac{\perp}{j-1} u_2 \) it follows that \( x_2 = \diamond u_3 \frac{\lozenge}{j} w_j \cdots \), where there exists \( u_0 \) such that \( u_0 \diamond u_1 = w_i \) and \( u_2 \diamond u_3 = w_j \). We obtain a contradiction because the second \( \frac{\lozenge}{j} \) of \( x_1 \) would be compatible with a \( \lozenge \) symbol of \( x_2 \).

• All the other cases when \( y = \diamond \) and \( z \in \{\frac{\lozenge}{j}, \frac{\perp}{j}, \frac{\ll}{j}, \frac{\under}{j}, \frac{\lozenge}{j}\} \) can be treated analogously to the above, and they all lead either to contradiction, either to the case when \( vv \) is a word that was already considered in the claims a,b,c,d.

By the analysis performed above it follows that claim \( e \) is correct. Thus, our proof is complete.

By a quite similar proof one can show that the following (more general) problem is \#P-complete.

**Problem 10.** Assume that \( p \) is a fixed natural number with \( p \geq 2 \). Given a partial word \( w \) over the alphabet \( V \), with \( |V| \geq 2 \), count the full words \( x \in V^* \) with \( 0 < |x| \leq |w| \) and \( x = v^p \) for some \( v \in V^* \) compatible with at least a factor of \( w \).

Again, according to [6, 7], this is an example of a hard counting problem that cannot be associated canonically with an NP-complete problem.

**Theorem 9.** Problem 10 is \#P-complete.

**Proof.** The proof goes on just like the proof of Theorem 8. The only difference is that the partial word we construct, in this case, starting from the input instance of Problem 3 is:

\[
w = (\frac{\lozenge}{j} w_1 \frac{\lozenge}{j})^p \frac{\lozenge}{j} k \frac{L+1}{j} \frac{s}{j} (\frac{\lozenge}{j} w_2 \frac{\lozenge}{j})^p \frac{\lozenge}{j} k \frac{L+2}{j} \frac{s}{j} \cdots (\frac{\lozenge}{j} w_k \frac{\lozenge}{j})^p \frac{\lozenge}{j} k \frac{L+k}{j} \frac{s}{j} \frac{2k^2 L^2 \lozenge}{j} v \lozenge \frac{L+1}{j}.
\]
Following arguments analogous to the above, the constructed word exhibits how Problem 3 can be Turing-reduced in polynomial time to Problem 10. Therefore Problem 10 is \#P-complete. □

5. A more general case

As we have already announced in the end of Section 3 we present now a solution for a general form of the problem of counting all the distinct full words that are compatible with factors of a given partial word.

In all the problems considered so far we have assumed that the set of symbols that can replace the hole is constant, and, usually, equal to the alphabet of the input words (and, for instance, this alphabet could be the alphabet of a Turing machine solving those problems). In the following we consider a more general framework: we assume that the set of symbols that can replace the hole is arbitrarily large, and is given as input, together with the encoding of a partial word over this alphabet. Note that this is a quite natural assumption: symbols/letters are usually represented as integers in the memory of a computer, thus, one can think that the ⋄ symbol can be actually replaced by any integer (or any data that is encoded in the same manner as symbols). Also, it is not unusual to consider arbitrary large alphabets (for instance, greater than the length of the input word) when one is interested in proving lower bounds (see [16], where a lower bound for the time needed to compute the edit distance was shown, under a similar assumption).

First we need a preliminary result: we note that a similar generalization of Problem 3 is a hard counting problem.

**Problem 11.** Given a natural number \(n\) and a list of partial words \(S = \{w_1, w_2, \ldots, w_k\}\) over an alphabet \(V\) with \(|V| = n\), each partial word having the same length \(L\), count the distinct words \(v \in V^L\) such that \(v\) is compatible with at least one of the partial words in \(S\).

We assume that \(n\) is given as a binary string (more precisely, \(n\) is given by its binary representation on exactly \(\lceil \log_2 n \rceil\) bits) and the symbols that appear in the partial words \(w_1, \ldots, w_k\) are encoded on exactly \(\lceil \log_2 n \rceil\) bits as the binary representation of the numbers less than or equal to \(n\) (for instance, ⋄ is encoded as the binary representation of the natural number 0 on \(\lceil \log_2 n \rceil\) bits, that is, \(0^{\lceil \log_2 n \rceil}\)). This means that the size of the input of this problem is \(O(kL \lceil \log_2 n \rceil)\); note that under the assumption that the set of symbols
compatible with the $\odot$-symbol is constant, used in the case of Problem 3, it followed that the size of the input of that problem was $O(kL)$.

We can easily show that Problem 11 is $\#P$-complete.

**Theorem 10.** Problem 11 is $\#P$-complete.

**Proof.** It is easy to see that this problem is in $\#P$. For the hardness part, note that Problem 3 is actually a particular case of Problem 11, in which $n$ is considered to be a constant. It is immediate that the general case is as hard as its particular case. In conclusion, Problem 11 is $\#P$-complete. $\square$

**Remark 1.** Note that even in the case when $n$ is not constant (for instance, when $n$ equals $f(k,L)$ for some linear function $f$) Problem 11 remains computationally hard. Indeed, assume that we have an efficient solution for it, and consider an instance of Problem 3. We can easily transform this instance of the basic problem by adding to the list several new partial words (more precisely, $(n-m)L$ new partial words, where $m$ is the list of symbols that were present in the initial words) such that any word, which contains at least one symbol that was not initially in the partial words of the list, is compatible with a word from the new list. Solving Problem 11, for this new list and, then, subtracting the number of words which contain at least a symbol that was not initially in the partial words of the list, leads to finding (efficiently) a solution of the initial problem. Therefore, Problem 11 is as hard as Problem 3, hence $\#P$-complete, even in the restricted cases mentioned above.

We stress that the hardness of such restricted versions of the general Problem 11 is important for the next proof.

We can now present the main result of this section.

**Problem 12.** Given a natural number $n$ and a partial word $w$ over an alphabet $V$ with $|V| = n$ count the distinct full words $v$ over the alphabet $V$ such that $v$ is compatible with at least one factor of $w$.

**Theorem 11.** Problem 12 is $\#P$-complete.

**Proof.** It is not hard to see how to construct a non-deterministic polynomial Turing machine having as many accepting paths as the number of words $x \in V^*$, with $0 < |x| \leq |w|$, compatible with a factor of $w$. Thus, Problem 12 is in $\#P$. We now show that it is also complete for this class.
In this respect, we propose a Turing-reduction from Problem 11. Assume that the function CountAll\((n,w)\) is the counting function defined in Problem 12, i.e., it counts the distinct full words \(v\) over the alphabet \(V\) such that \(v\) is compatible with at least one factor of \(w\). Further, following the discussion made in Remark 1, we consider an input instance of Problem 11: the number \(n = k + L + 2\), and the list \(S = \{w_1, w_2, \ldots, w_k\}\), with \(k > 3\), containing partial words of length \(L\) with \(L > 2\) over the alphabet \(\{0, 1\}\); we can assume, without loss of generality, that all these words start with a symbol different from \(\diamond\) (we can add a starting symbol 0 to all the partial words, to ensure that this assumption holds, and increase \(n\) and \(L\) by 1, accordingly).

Next, consider the alphabets \(V_1 = \{\#_1, \#_2, \ldots, \#_L, \#_{L+1}\}\) and \(V_2 = \{\$, \#_1, \ldots, \#_k\}\). Let \(V = V_1 \cup V_2\). For a partial word \(u = a_1a_2 \cdots a_L\), of length \(L\), we define \(#(u) = #_1a_1#_2a_2 \cdots #La_L#_{L+1}\).

Finally, we define the partial word \(w\) by:
\[
 w = \diamond\#_2\#^{2L-3}\#_{L+1}\$\, #(w_1)$\#(w_2)$\#(w_k)$\#_1\#^{2L-3}\#_L.
\]

We show that if we are able to compute efficiently the number of distinct full words over \(V\) compatible with factors of \(w\), then we are able to compute efficiently the number of full words over \(V\) compatible with at least one word from the list \(S\), denoted in the following by \(N_{n,S}\).

Now let us show how the number of distinct full words that are contained in the partial word \(w\) can be computed:

- All the full words over \(V\) of length at most \(2L - 3\) are compatible with a factor \(\diamond^\ell\), with \(\ell \leq 2L - 3\), of \(w\). The number of such words is \(\sum_{1 \leq \ell \leq 2L-3} n^\ell\). Clearly, this number can be computed in polynomial time.

- A full word of length \(2L - 2\) which is compatible with a factor of \(w\) is for sure compatible with one of the partial words \(\diamond\#_2\#^{2L-4}, \#_2\#^{2L-3}, \#^{2L-3}\#_{L+1}, \#_1\#^{2L-3}, \#^{2L-3}\#_L\), or with a factor of \(w\) that contains a symbol from \(V_2\), as well. In the latter case, the full word is of the form \(u\$iv\), for some \(i \in \{1, \ldots, k\}\), \(u, v \in V^*\) and \(|u| + |v| = 2L - 3\). It follows that it contains either \(#_{L+1}\$i\) or \$i#_1\); in both cases, there exists exactly one factor of \(w\) which is compatible with this word: the factor that contains \$i\) on exactly the same position as the full word does. Consequently, to count the full words of length \(2L - 2\) compatible with factors of \(w\) we proceed as follows. First, for all \(i \in \{1, \ldots, n\}\) and
all the factors of $w$ of length $2L - 2$ that contain $\$; we compute the number of full words compatible with that factor of $w$ (that is, $|V|$ to the number of holes contained in that factor) and subtract from it the number of full words that are also compatible with one of the partial words $\oplus#_2^{|2L-4|}$, $\oplus#_2^{|2L-3|}$, $\oplus#_1^{|2L-3|}$, or $\oplus#_1^{|2L-3|}$#L (as these words will be added to the total number of words of length $2L - 2$ compatible with factors of $w$ later). The result is stored as $N_{i,2L-2}$. Then we compute $\sum_{i \in \{1, \ldots, k\}} N_{i,2L-2}$ and add to this sum the number of words that are compatible with one of the partial words $\oplus#_2^{|2L-4|}$, $\oplus#_2^{|2L-3|}$#L+1, $\oplus#_1^{|2L-3|}$#L+1, or $\oplus#_1^{|2L-3|}$#L; let $N_{2L-2}$ denote the value computed by this procedure. Clearly, all the steps described above can be completed in deterministic polynomial time. To conclude, $N_{2L-2}$, the number of words of length $2L - 2$ compatible with factors of $w$, can be computed in polynomial time.

- A full word of length $2L - 1$ which is compatible with a factor of $w$ is for sure compatible with one of the partial words $\oplus#_2^{|2L-3|}$, $\oplus#_2^{|2L-3|}$#L+1, $\oplus#_1^{|2L-3|}$#L, or with a factor of $w$ that contains a symbol from $V_2$, as well. Similarly to the above, in the last case the full word has the form $u\$;v, for some $i \in \{1, \ldots, k\}$, $u, v \in V^*$ and $|u| + |v| = 2L - 2$, and we easily obtain that it contains either $#L+1\$; or $\$;#1; in both cases, it follows that there exists exactly one factor of $w$ which is compatible with this word, namely the one having $\$; on exactly the same position as in the full word. Consequently, to count the full words of length $2L - 1$ compatible with factors of $w$ we proceed in the exact manner that we used in the case of the words of length $2L - 2$. First, for all $i \in \{1, \ldots, n\}$ and all the factors of $w$ of length $2L - 1$ that contain $\$$; we compute the number of full words that are compatible with that factor of $w$ (i.e., $|V|$ to the number of holes contained in the factor) and subtract from it the number of words that are also compatible with one of the partial words $\oplus#_2^{|2L-3|}$, $\oplus#_2^{|2L-3|}$#L+1, or $\oplus#_1^{|2L-3|}$#L. The result is the number $N_{i,2L-1}$. Then, we compute $\sum_{i \in \{1, \ldots, k\}} N_{i,2L-1}$ and add to this sum the number of words that are compatible with one of the partial words $\oplus#_2^{|2L-3|}$, $\oplus#_2^{|2L-3|}$#L+1, or $\oplus#_1^{|2L-3|}$#L; let $N_{2L-1}$ denote the value computed in this way. All the steps described above can be, clearly, completed in deterministic polynomial time. In conclusion, $N_{2L-1}$, the number of words of length $2L - 1$ compatible with factors of $w$, can be computed in polynomial time.
A full word of length $2L$ which is compatible with a factor of $w$ is for sure compatible with the partial word $\diamond \#_2 \diamond^{2L-3} \#_{L+1}$, with a factor of length $2L$ of $w$ which starts with $\#_1$, or with a factor of $w$ that contains a symbol from $V_2$, as well. Once again, in the last case the full word has the form $u\$_i v$, for some $i \in \{1, \ldots, k\}$, $u, v \in V^*$ and $|u| + |v| = 2L - 1$, and it follows that it must contain either $\#_{L+1} \$_i$ or $\$_i \#_1$; in both cases, we get that the factor of $w$ which is compatible with this word is uniquely determined. Also, in the case when the full word is compatible with a factor of $w$ of length $2L$ which starts with $\#_1$ then it is not compatible with the factor $\diamond \#_2 \diamond^{2L-3} \#_{L+1}$ or with a factor that contains a symbol from $V_2$. Therefore, in order to count the full words of length $2L$ compatible with factors of $w$ we can sum up the number of words that are compatible with a factor of $w$ that contain a symbol from $V_2$, and the number of words that are compatible with factors of $w$ that start with $\#_1$. To compute the first number, denoted in the following $N_{2L}$, we proceed as in the previous cases: for all $i \in \{1, \ldots, n\}$ and all the factors of $w$, of length $2L$, that contain $\$_i$, we compute the number of full words that are compatible with that factor of $w$ and subtract from it the number of words that are also compatible with $\diamond \#_2 \diamond^{2L-3} \#_{L+1}$; then we sum up all these numbers and add the number of words that are compatible with $\diamond \#_2 \diamond^{2L-3} \#_{L+1}$. Clearly, $N_{2L}$ can be computed in polynomial time. The second number is exactly the number $N_{n, S}$ of full words over $V$ that are compatible with at least one of the partial words from the list $S$.

A full word of length $2L + 1$ compatible with a factor of $w$ can only be compatible with a factor of length $2L + 1$ of $w$ which starts with $\#_1$, or with a factor of $w$ that contains a symbol from $V_2$. In the latter case, the full word has the form $u\$_i v$, for some $i \in \{1, \ldots, k\}$, $u, v \in V^*$ and $|u| + |v| = 2L$, and it follows that the factor of $w$ which is compatible with this word is uniquely determined. Thus, to count the full words of length $2L$ compatible with factors of $w$ we can sum up the number of words that are compatible with a factor of $w$ that contain a symbol from $V_2$, and the number of words that are compatible with factors of $w$, of length $2L + 1$, that start with $\#_1$. The first number, denoted in the following $N_{2L+1}$, can be computed in polynomial time, as we have discussed already above. The second number equals the number
\( N_{n,S} \) of full words over \( V \) which are compatible with at least one of the partial words from the list \( S \).

- A full word of length \( M > 2L + 1 \) compatible with a factor of \( w \) can only be compatible with a factor of \( w \) that contains a symbol from \( V_2 \). Such a word has the form \( u\$_i v \), for some \( i \in \{1, \ldots, k\} \), \( u, v \in V^* \) and \( |u| + |v| = M - 1 \), and it follows that the factor of \( w \) which is compatible with this word is uniquely determined. As discussed above, the number of such words, denoted \( N_M \), can be computed in polynomial time, for all \( M \leq |w| \).

According to the above, the value of \( \text{CountAll}(n, w) \) is equal to

\[
\sum_{1 \leq \ell \leq 2L - 3} n^\ell + \sum_{i \in \{2L - 3, 2L - 2, \ldots, |w|\}} N_i + 2N_{n,S}.
\]

Therefore,

\[
N_{n,S} = \left( \text{CountAll}(n, w) - \left( \sum_{1 \leq \ell \leq 2L - 3} n^\ell \right) + \left( \sum_{i \in \{2L - 3, 2L - 2, \ldots, |w|\}} N_i \right) \right) / 2.
\]

This shows that Problem 11 can be Turing-reduced in polynomial time to Problem 12. Therefore, Problem 12 is \#P-complete, as well.

The assumption that the alphabet of the symbols that can replace the hole can be arbitrarily large was essentially used in the above proof, in the construction of the partial word \( w \). We were not able to show a similar bound under more restrictive assumptions.

6. An upper bound

Usually, the approaches for solving \#P-complete problems were based on (probabilistic) approximation algorithms; we recall the results in [17, 18, 19, 20] as references on how the problems from \#P can be theoretically solved by such algorithms. On the other hand, there are several approaches on designing exact (yet, exponential) algorithms that solve counting problems [21, 22]. For instance, in [22] are presented exact algorithms solving several hard counting complete problems (most important, the counting variant of the exact satisfiability problem and the problem of counting all the perfect matchings); these algorithms, working in exponential time and, some
of them, polynomial space, are less complex than the exhaustive search, and seem to be a good alternative for practically solving such problems. Here, we take some steps in the same direction. We propose a nontrivial exact exponential algorithm, working in exponential time and polynomial space, for computing the answer to Problem 3. This algorithm seems even more interesting as it can be easily adapted to solve most of the other counting problems discussed in this paper, more efficiently than by the naive strategy of exhaustively exploring the solutions space. It is also worth noting that our algorithm, similarly to some of the algorithms in [22] and to the classical Ryser formula for the permanent of a matrix [21], is based on the inclusion-exclusion principle.

Also, before starting the presentation of our algorithm, let us mention that whenever we discuss the complexity of an algorithm we use as model of computation the unit cost RAM model [23].

Let the list \( S = \{w_1, w_2, \ldots, w_k\} \), containing partial words of length \( L \) over an alphabet \( V \), define an instance of Problem 3. In the following we make the assumption that \( V \) is the minimal alphabet such that all the partial words in \( S \) are contained in \((V \cup \{\diamond\})^*\); it is not hard to see that in this case \(|V| \leq kL\).

The algorithm we propose is based on the Inclusion-Exclusion Principle. In order to apply this method we define the sets:

\[
B_i = \{v | v \in V^L \text{ and } w_i \uparrow v\}, \quad \text{for } i \in \{1, \ldots, k\}
\]

That is, the set \( B_i \) contains all the full words compatible with the partial word \( w_i \).

Clearly, solving Problem 3 is equivalent to computing the cardinality of set \( B_1 \cup B_2 \cup \cdots \cup B_k \). Applying the inclusion-exclusion principle we have:

\[
|B_1 \cup B_2 \cup \cdots \cup B_k| = |B_1| + \cdots + |B_k| - |B_1 \cap B_2| - |B_1 \cap B_3| - \cdots + (-1)^{k+1}|B_1 \cap B_2 \cap \cdots \cap B_k|
\]

It remains to show how we can compute the cardinality of each term from the above formula; more precisely, we want to compute the cardinality of an arbitrary intersection of some of the sets \( B_1, B_2, \ldots, B_k \).

Let us suppose that we want to compute \(|B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_p}|\), where \( i_t \in \{1, \ldots, k\} \), for all \( t \leq p \), and \( i_j \neq i_\ell \) for all \( j \neq \ell \).

First, we define the matrix \( Aux \) with \(|V| + 1 \) rows and \( L \) columns having integer elements. The value \( Aux[x][\ell] \) stores the number of times the symbol \( x \), with \( x \in V \cup \{\diamond\} \), appears on the \( \ell^{th} \) position of the partial words.
$w_{i_1}, \ldots, w_{i_p}$, for each $\ell \in \{1, \ldots, L\}$. We observe that we can compute the cardinality of the set $B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_p}$, using the matrix $ Aux $, as follows:

i. The cardinality is 0 if and only if there exists a position $\ell \in \{1, \ldots, L\}$ and at least two different symbols $x, y \neq \diamond$, such that $ Aux[x][\ell] \geq 1 $ and $ Aux[y][\ell] \geq 1 $.

ii. Otherwise the cardinality is $|V|^m$, where $m$ is the number of positions $\ell, \ell \in \{1, \ldots, L\}$, for which $ Aux[\diamond][\ell] = p $.

In order to implement the above computation efficiently we also use an array $ Counter $ of size $L$ with integer elements. More precisely, $ Counter[\ell] $ denotes the number of distinct symbols $x$, with $x \neq \diamond$, such that $ Aux[x][\ell] \neq 0 $. Therefore, we can reformulate the case (i.) from the above list, as follows:

- The cardinality of the set $B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_p}$ equals 0 if and only if there exists a position $\ell \in \{1, \ldots, L\}$ such that $ Counter[\ell] \geq 2 $.

A remark, quite important for our strategy, is that if we add (or, respectively, remove) a set $B_i$ to (from) the intersection, we can update both the matrix $ Aux $ and the array $ Counter $ in $O(L)$ time.

More precisely, let us assume $B_i$ is added to the current intersection. Then, for each $\ell \in \{1, \ldots, L\}$ we perform the following steps:

- If $w_i[\ell] \neq \diamond$ and $ Aux[w_i][\ell] = 0 $, then set $ Counter[\ell] = Counter[\ell] + 1 $;
- In all the cases set $ Aux[w_i][\ell] + 1 $.

Indeed, when we intersect $B_{i_1} \cap \cdots \cap B_{i_p}$ with new set $B_i$, that corresponds to $w_i$, we have two cases for each $\ell \leq L$: $w_i[\ell] = \diamond$ or $w_i[\ell] \neq \diamond$. In the first case we just have to increase $ Aux[\diamond][\ell] $ with 1, as a new $\diamond$ symbol appeared at position $\ell$ of the partial words corresponding to the intersected sets. In the second case the discussion splits in two new cases. When $w_i[\ell]$ appeared already in one of the partial words associated with the intersected sets we just have to increase $ Aux[w_i][\ell] $ by 1, but the number of different symbols that appear in all the partial words at position $\ell$ remains the same (that is, $ Counter[\ell] $ remains unchanged). When $w_i[\ell]$ did not appear already in one of the partial words associated with the intersected sets we increase $ Aux[w_i][\ell] $ by 1, and we also increase by 1 the number of different symbols that appear in all the partial words at position $\ell$ remains the same (that is, $ Counter[\ell] $ is increased by 1).
By similar reasons, if $B_i$ is removed from the current intersection we have, for each $\ell \in \{1, \ldots, L\}$:

- If $w_i[\ell] \neq \Diamond$ and $\text{Aux}[w_i][\ell] = 1$, then set $\text{Counter}[\ell] = \text{Counter}[\ell] - 1$ (clearly, $\text{Counter}[\ell]$ is at least 1 before the removal of $B_i$);
- In all the cases set $\text{Aux}[w_i][\ell] = \text{Aux}[w_i][\ell] - 1$ (clearly, $\text{Aux}[w_i][\ell]$ is at least 1 before the removal of $B_i$).

A more careful analysis of the possible values that the cardinality of a set $B_{i_1} \cap B_{i_2} \cap \cdots \cap B_{i_p}$ may take shows that they can be represented as a monomial in $|V|$ (it can be either 0 or, respectively, $|V|^t$, for some $t \in \{0, \ldots, L\}$). Thus, we are interested in computing only the degree of this monomial. Moreover, we can store a polynomial $P_S$ of degree at most $L$ (i.e., an array of $L$ integer coefficients), and, instead of computing each time the cardinality of the current intersection of sets and updating the result, we could add or subtract (depending on the sign that the current intersection of sets has in the formula given by the Inclusion-Exclusion Principle) the monomial computed for the currently analysed intersection to $P_S$; this can be done in $O(1)$ time as it consists, at most, in increasing or decreasing with 1 one of the coefficients of the polynomial $P_S$, namely the coefficient that corresponds to the degree of the current monomial. Then, when we finished analysing all the intersections, we just have to compute the value of $P_S(|V|)$.

On the other hand, the alphabet $V$ may also be seen as a variable. That is, we may be interested in solving Problem 11 for the same input set of partial words, but for different sizes of $V$. In this case, once we compute the coefficients of $P_S$ we are able to find the answer to Problem 11, for every alphabet $W$ which includes the minimal alphabet $V$ of words of $S$. In other words, we may increase the size of the alphabet replacing the $\Diamond$-symbol but the initial list $S$ of partial words remains unchanged. The key remark is that the coefficients of the polynomial $P_S$ depend only on the list $S$, and not on $V$, as long as all the alphabet that can replace the $\Diamond$-symbol includes all the symbols present in the initial words. In conclusion, once we know the coefficients of this polynomial, we can compute efficiently (in $O(L)$ time) the result to Problem 11, for the list of words $S$ and different alphabets that include $V$.

To obtain the overall time complexity of computing the aforementioned polynomial, we proceed as follows. Let us assume that we could consider the subsets of $\{1, \ldots, k\}$ in the following order $S_1$, $S_2$, $\ldots$, $S_{2^k}$, such that
every two consecutive subsets from this list differ by exactly one element. We use this list to compute the cardinality of the intersections in the inclusion-exclusion-principle-formula as follows: first we intersect the sets with indexes in \( S_1 \), then the sets indicated by \( S_2 \), and so on. In this manner, we could use the matrix \( Aux \) to compute the cardinality for each such intersection in time \( O(L) \), as we have already explained.

Note that the list above can be obtained using the binary reflected Gray code of size \( k \) ([24]). The elements of this code are the \( 2^k \), pairwise different, bit-strings \( g_1, \ldots, g_{2^k} \), each having \( k \) bits; we have in general \( g_i = i \oplus \lfloor i/2 \rfloor \), and, in particular, \( g_1 = 0^k \) and \( g_{2^k} = 10^{k-1} \). Each two consecutive elements \( g_i \) and \( g_{i+1} \) differ only by one bit. The elements of the Gray code of size \( k \) can be computed in \( O(2^k) \) time (as we assume that the computing model, unit cost RAM, permits the computation of bitwise-operations, like shifting and exclusive-or, denoted here by \( \oplus \), on numbers with \( k \) bits in constant time). We say that the sets \( S_{i_1}, \ldots, S_{i_l} \) are indicated by \( g_i \) if and only if the bits of \( g_i \) that are equal to 1 are found at the positions \( i_1, \ldots, i_l \).

Summarizing all the considerations described above, we obtain Algorithm 1 that computes the cardinality \( |B_1 \cup \cdots \cup B_k| \), as a polynomial in \( |V| \).

The time complexity of Algorithm 1 is \( O(2^k L) \). The space complexity is bounded by \( O(k^2 L) \). It is interesting to note that if we extend the size of the alphabet that can replace the holes, the time complexity of the algorithm remains just the same. Moreover, once we have computed the polynomial \( P_S \) (which depends only on the partial words from \( S \), and the symbols that already exist in these words), if we update the alphabet we need only to compute the value \( P_S(m) \), where \( m \) is the cardinality of the new alphabet.

A heuristic that can be used to make the algorithm more efficient is to eliminate all the full words from \( S \). Clearly, such a full word should be added to the final result if and only if it is not compatible with any of the other partial words from the list; this property can be checked, for every full word of the list, in \( O(k L) \) time. In this way we ensure that the overall complexity of the algorithm is less than \( O(\max(k^2, 2^h)L) \), where \( h \) is the number of partial words from the list \( S \) that contain holes.

**Theorem 12.** Problem 3 can be solved in time \( O(\max(2^h, k^2)L) \), where \( h \) is the number of partial words of the list \( S \) that contain holes.

It is worth making a comparison between our algorithm and an exhaustive search. In the naive approach we can generate, in order, all the full words in
Algorithm 1 \textit{Count}(S)
Input: a list of partial words \( S = \{w_1, w_2, \ldots, w_k\} \) over the alphabet \( V \) with \( |V| \geq 2 \) all having the same length \( L \)
Returns: the number of full words compatible with at least one partial word from \( S \)

1: Set \( P_S(X) = 0 \), where \( P_S \) is a polynomial of maximum degree \( L \) in the variable \( X \);
2: Allocate memory for the matrix \( Aux \) with \( |V| + 1 \) rows and \( L \) columns, and initialize all its elements with 0;
3: Allocate memory for the array \( Counter \) with \( L \) positions, and initialize all its elements with 0;
4: \textbf{for} \( i \in \{2, 3, \ldots, 2^k\} \) \textbf{do}
5: \hspace{1em} Let \( g_i \) be the \( i^{th} \) element of the Gray code of length \( k \); \( g_i = i \oplus \lfloor i/2 \rfloor \)
6: \hspace{1em} Let \( \ell \) be the position of the bit which is different between \( g_i \) and \( g_{i-1} \)
7: \hspace{1em} Insert (or, respectively extract) the set \( B_\ell \) in (respectively, from) the current intersection if the bit on position \( \ell \) of \( g_i \) was 0 (respectively, if that bit was 1); Update the matrix \( Aux \) and the array \( Counter \) accordingly (as we have described above); Compute the number \( t \) such that cardinality of the intersection of the sets indicated by \( g_i \) is \( |V|^t \);
8: \hspace{1em} Add or subtract the monomial \( X^t \) to the polynomial \( P_S \) (according to the sign of the current intersection in the formula given by the inclusion-exclusion principle);
9: \textbf{end for}
10: Return the value \( P_S(n) \), where \( n \) is the size of an alphabet that includes all the symbols of the partial words from \( S \).
V^L$, and count how many of them are compatible with a word in $S$. Clearly, such an approach can be implemented in time $O(kL|V|^L)$. Therefore, we can state that, intuitively, our approach works better than the exhaustive search whenever $L$ is not much smaller than $h$. Recall, though, that our approach has the nice feature that once $P_S$ computed we can change the size of the alphabet and compute in $O(L)$ time the number of full words over the new alphabet compatible with at least one word of $S$; clearly, this is not the case for the naive approach. However, it seems interesting to us also to study algorithmic solutions for this problem also for the case when $L$ is much smaller than $h$, for instance $L \in O(\log h)$. Finally, it is worth noting that also in the case of counting the solutions of the Exact Satisfiability problem, in [22], an algorithm with the running time exponential in the number of clauses of a formula was obtained, differently from other algorithms solving this problem, which had a running time exponential in the number of variables.

Upper bounds for solving all the problems discussed in this paper follow from Theorem 12; we only give several examples of such upper bounds, as the others can be shown quite similarly. In the case of Problem 5, for instance, we form a list containing the factors of length $L$ of $w$, and we apply the above algorithm to this list; the time complexity of this solution is clearly $O(n 2^n)$. Counting all the full words that are compatible with factors of a partial word takes $O(n 2^n)$ time, since we can count separately the full words of length less than the length of the input word that are compatible with factors of this word, and then sum up the results. Also, Problems 9 and 10 can be solved in time $O(n 2^n)$: we identify the factors of the input word that are compatible with a square (respectively, a $k$-repetition), separate them into several lists according to their length, and then solve the problem for these lists as discussed before.

7. Conclusions

An immediate continuation of the results presented in this paper would be to solve Problem 12 for the case when the alphabet of symbols that can replace the holes is constant (for instance, it coincides with the alphabet of the input partial word). Also, any improvement of the upper bounds derived from the algorithm presented in Section 6 would be interesting.

On the other hand, there are several other counting problems for partial words (for instance, counting the bordered or unbordered partial words of a given length, [3]) for which no polynomial algorithms are known. Of course,
an investigation of such problems, following the one performed in this pa-
per, seems appealing: are they \#P-hard/complete, or can they be solved
efficiently? Of course, in the case when such a problem is proven to be hard,
one could be interested in finding non-trivial upper bounds for the running
time of algorithms solving it.

Acknowledgements

Part of this paper was written during the stay of Florin Manea at the Depart-
ment of Knowledge and Language Engineering, Faculty of Computer Science,
Otto-von-Guericke University in Magdeburg, Germany, as a Research Fellow
of the Alexander von Humboldt Foundation, whose support is gratefully ac-
knowledged. Florin Manea thanks Professor Jürgen Dassow for his advices
in writing this paper.
The authors would also like to thank to the anonymous referees for their very
useful comments and suggestions.

References

[1] F. Manea, C. Tiseanu, Hard counting problems for partial words, in:

[2] J. Berstel, L. Boasson, Partial words and a theorem of Fine and Wilf,

[3] F. Blanchet-Sadri, Algorithmic Combinatorics on Partial Words, Chap-


conjecture on overlaps in partial words using periodicity algorithms, in:
2009, pp. 188–199.


35


