Normal, Abby Normal, Prefix Normal

Péter Burcsi[1], Gabriele Fici[2], Zsuzsanna Lipták[3], Frank Ruskey[4], and Joe Sawada[5]

[1] Dept. of Computer Algebra, Eötvös Loránd Univ., Budapest, Hungary, bupe@compalg.inf.elte.hu
[2] Dip. di Matematica e Informatica, University of Palermo, Italy, gabriele.fici@math.unipa.it
[3] Dip. di Informatica, University of Verona, Italy, zsuzsanna.liptak@univr.it
[4] Dept. of Computer Science, University of Victoria, Canada, ruskey@cs.uvic.ca
[5] School of Computer Science, University of Guelph, Canada, jsawada@uoguelph.ca

Abstract. A prefix normal word is a binary word with the property that no substring has more 1s than the prefix of the same length. This class of words is important in the context of binary jumbled pattern matching. In this paper we present results about the number $pnw(n)$ of prefix normal words of length n, showing that $pnw(n) = \Omega\left(2^{n-c\sqrt{n \ln n}}\right)$ for some c and $pnw(n) = O\left(\frac{2^n (\ln n)^2}{n}\right)$. We introduce efficient algorithms for testing the prefix normal property and a "mechanical algorithm" for computing prefix normal forms. We also include games which can be played with prefix normal words. In these games Alice wishes to stay normal but Bob wants to drive her "abnormal" – we discuss which parameter settings allow Alice to succeed.

Keywords: prefix normal words, binary jumbled pattern matching, normal forms, enumeration, membership testing, binary languages

1 Introduction

Consider the binary word $w = 10100110110001110010$. Does it have a substring of length 11 containing exactly 5 ones? In Fig. 1 the word w is represented by the black line (go up and right for a 1, down and right for a 0), while the grid points within the area between the two lighter lines form the *Parikh set* of w: the set of vectors (x, y) s.t. some substring of w contains exactly x ones and y zeros. Since the point $(5, 6)$ lies within the area bounded by the two lighter lines, we see that the answer to our question is 'yes'. (Don't worry, more detailed explanation will follow soon.) Now, this paper is about the lighter lines, called *prefix normal words.*

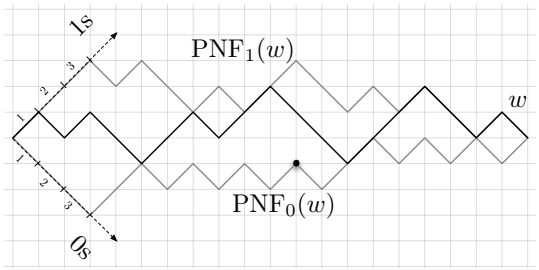

Fig. 1. The word $w = 10100110110001110010$ (dark line), its prefix normal forms $\mathrm{PNF}_1(w) = 11101001011001010010$ and $\mathrm{PNF}_0(w) = 00011010101011010101$ (lighter lines); the region between the two is the Parikh set of w; e.g. w has a substring containing 5 ones and 6 zeros (black dot). Note that the axes are rotated by 45 degrees clockwise.

Prefix normal words: A binary word w is called *prefix normal* (with respect to 1) if no substring of w has more 1s then the prefix of the same length[6]. For example, 110101101100100 is not prefix normal because it has a substring of length 5 with 4 ones, while the prefix of length 5 has only 3 ones. In [14] it was shown that to every word w, one can assign two prefix normal words, the *prefix normal forms* (PNF) of w (w.r.t. 1 and w.r.t. 0), and that these are precisely the lines bounding w's Parikh set from above (w.r.t. 1) resp. from below (w.r.t. 0), interpreted as binary words.

Prefix normal games: Before we further elaborate on the connection between the initial problem and prefix normal words, let's see how well you have understood the definition. To this end, we define a two-player game. At the start of the game Alice and Bob have n free positions. Alice moves first: she picks a position and sets it to 0 or 1. Then in alternating moves, they pick an empty position and set it. The game ends after n moves. Alice wins if and only if the resulting binary word is prefix normal.

Example 1. Here is an example run. We have $n = 5$. Alice sets the first bit to 1, then Bob sets the second bit to 0. Now Alice sets the 4th bit to 0, and she has won, since whichever position Bob chooses, she will set the remaining position to 0, thus ensuring that the word is prefix normal.

1.	*start*	_	_	_	_	_	3.	*Bob*	1	0	_	_	_
2.	*Alice*	1	_	_	_	_	4.	*Alice*	1	0	_	0	_

The solution to the following exercise can be found in Section 6.

Exercise 1. Find the maximum n such that Alice has a winning strategy.

Binary Jumbled Pattern Matching: The problem of deciding whether a particular pair (x, y) lies within the Parikh set of a word w is known as *binary jumbled pattern matching*. There has been much interest recently in the *indexed version*, where an index for the Parikh set is created in a preprocessing step, which can then be used to answer queries fast. The Parikh set can be represented in linear space due to the *interval property* of binary strings: If w has k-length substrings with x_1 resp. x_2 ones, where $x_1 < x_2$, then it also has a k-length substring with y ones, for every $x_1 \leq y \leq x_2$ (folklore). Thus the Parikh set can be represented by storing, for every $1 \leq k \leq |w|$, the minimum and maximum number of 1s in a substring of length k. Much recent research has focused on how to compute these numbers efficiently [2, 10, 12, 15, 16, 20, 21]. The problem has also been extended to graphs and trees [11, 15], to the streaming model [19], and to approximate indexes [12]. There is also interest in the non-binary variant [9, 10, 18]. A closely related problem is that of Parikh fingerprints [1]. Applications in computational biology include SNP discovery, alignment, gene clusters, pattern discovery, and mass spectrometry data interpretation [3, 4, 5, 13, 23].

The current best construction algorithms for the linear size index for binary jumbled pattern matching run in $O(n^2/\log n)$ time [7, 20], for a word w of length n, with some improvements for special cases (compressible strings [2, 15], bit-parallel operations [16, 21])[7]. As we will see later, computing the prefix normal forms is equivalent to creating an index for the Parikh set of w. Currently, we know no faster computation algorithms for the prefix normal forms than already exist for the linear-size index. However, should better algorithms be discovered, these would immediately carry over to the problem of indexed binary jumbled pattern matching.

[6] When not specified, we mean prefix normal w.r.t. 1.

[7] Very recently, an algorithm with running time $n^2/2^{\Omega(\log n/\log\log n)^{1/2}}$ was presented [17].

Testing: It turns out that even *testing* whether a given word is prefix normal is a nontrivial task. We can of course compute w's prefix normal form, in $O(n^2/\text{polylog } n)$ time using one of the above algorithms: obviously w is prefix normal if and only if $w = \text{PNF}(w)$. In [8], we gave a generating algorithm for prefix normal words, which exhaustively lists all prefix normal words of a fixed length. The algorithm was based on the fact that prefix normal words are a bubble language, a recently introduced class of binary languages [24, 26]. As a subroutine of our algorithm, we gave a linear time test for words which are obtained from a prefix normal word via a certain operation. In Section 7, we present an algorithm to test whether an *arbitrary* word is prefix normal, based on similar ideas. Our algorithm is quadratic in the worst case but we believe it performs much better than other algorithms once some simple cases have been removed.

We further demonstrate how using several simple linear time tests can be used as a filtering step, and conjecture, based on experimental evidence, that these lead to expected $O(n)$ time algorithms. But first the reader is kindly invited to try for herself.

Exercise 2. Decide whether the word 111010100110110011 is prefix normal.

Enumerating: Another very interesting and challenging problem is the enumeration of prefix normal words. It turns out that even though the number of prefix normal words grows exponentially, the fraction of these words within all binary words goes to 0 as n goes to infinity. In Sections 3 to 5, we present both asymptotic and exact results for prefix normal words, including generating functions for special classes and counting extensions for particular words. Some of the proofs in this part of the paper are rather technical: they will be available in the full version.

Mechanical algorithm design: We contribute to the area of mechanical algorithm design by presenting an algorithm for computing the Parikh set which uses the new *sandbeach technique*, a technique we believe will be useful in many other applications (Sec. 7).

We would like to point out that prefix normal words, albeit similar in name, are not to be confused with so-called *Abby Normal* (a.k.a. *abnormal* or *AB normal*), words, or rather, brains, introduced in [6].— And now it is time to wish you, the reader, as much fun in reading our paper as we had in writing it!

2 Prefix normal words

A *binary word* (or *string*) $w = w_1 \cdots w_n$ over $\Sigma = \{0, 1\}$ is a finite sequence of elements from Σ. Its length n is denoted by $|w|$. For any $1 \leq i \leq |w|$, the i-th symbol of a word w is denoted by w_i. We denote by Σ^n the words over Σ of length n, and by $\Sigma^* = \cup_{n \geq 0} \Sigma^n$ the set of finite words over Σ. The empty word is denoted by ε. Let $w \in \Sigma^*$. If $w = uv$ for some $u, v \in \Sigma^*$, we say that u is a *prefix* of w and v is a *suffix* of w. A *substring* of w is a prefix of a suffix of w. A *binary language* is any subset $\mathcal{L}$ of Σ^*. We denote by $|w|_c$ the number of occurrences in w of character $c \in \{0, 1\}$; $|w|_1$ is called the *density* of w.

Let $w \in \Sigma^*$. For $i = 0, \ldots, n$, we set $P(w, i) = |w_1 \cdots w_i|_1$, the number of 1s in the i-length prefix of w, and $F(w, i) = \max\{|u|_1 : u \text{ is a substring of } w \text{ and } |u| = i\}$, the maximum number of 1s over all substrings of length i.

Prefix normal words, prefix normal equivalence and prefix normal form were introduced in [14]. A word $w \in \{0, 1\}^*$ is *prefix normal* (w.r.t. 1) if, for all $1 \leq i \leq |w|$, $F(w, i) = P(w, i)$. In other words, a word is prefix normal if no substring contains more 1s than the prefix of the same length.

Example 2. We give all 23 prefix normal words of length $n = 6$:
000000, 100000, 100001, 100010, 100100, 101000, 101001, 101010, 110000, 110001, 110010, 110011, 110100, 110101, 110110, 111000, 111001, 111010, 111011, 111100, 111101, 111110, 111111.

Two words w, w' are *prefix normal equivalent* (w.r.t. 1) if and only if $F(w, i) = F(w', i)$ for all i. Given $w \in \Sigma^*$, the *prefix normal form* (w.r.t. 1) of w, $\mathrm{PNF}(w) = \mathrm{PNF}_1(w)$, is the unique prefix normal word w' which is prefix normal equivalent (w.r.t. 1) to w. Prefix normality w.r.t. 0, prefix normal equivalence w.r.t. 0, and $\mathrm{PNF}_0(w)$ are defined analogously. When not stated explicitly, we are referring to the functions w.r.t. 1. For example, the words 0000111 and 1110000 are prefix normal equivalent both w.r.t. 0 and 1. See [8, 14] for more examples.

In Fig. 1, we see an example string w and its prefix normal forms. The interval property (see Introduction) can be graphically interpreted as vertical lines. The vertical line through point $(5, 6)$ represents length-11 substrings: the grid points within the enclosed area are $(7, 4), (6, 5)$, and $(5, 6)$, so all length-11 substrings have between 7 and 5 ones. We can interpret, for each length k, the intersection of the kth vertical line with the top grey line as the maximum number of 1s, and with the bottom grey line as the minimum number of 1s. Now it is easy to see that, passing from k to $k + 1$, this maximum, $F_1(w, \cdot)$, can either remain the same or increase by one. This means that the top grey line allows an interpretation as a binary word. A similar interpretation applies to the bottom line and prefix normal words w.r.t 0.

It should now be clear, also graphically, that the maximum number of 1s for a substring of length k, $F(w, k)$, is precisely the number of 1s in the k-length prefix of $\mathrm{PNF}_1(w)$ (the upper grey line); and similarly for the maximal number of 0s (equivalently, the minimal number of 1s) and $\mathrm{PNF}_0(w, k)$ (the lower grey line). Moreover, these values can be obtained in constant time with constant-time rank-operations [15, 22].

We list a few properties of prefix normal words that will be useful later.

Lemma 1 (Properties of prefix normal words [14]).

1. *Every prefix of a prefix normal word is also prefix normal.*
2. *If w is prefix normal, then $w0$ is also prefix normal.*
3. *Given w of length n, it can be decided in $O(n^2)$ time whether w is prefix normal.*

We denote the language of prefix normal words by $\mathcal{L}_{\mathrm{PN}}$, the number of prefix normal words of length n by $pnw(n)$, and the number of prefix normal words of length n and density d by $pnw(n, d)$. The first few values of the sequence $pnw(n)$ are listed in [25].

3 Asymptotic bounds on the number of prefix normal words

We give lower and upper bounds on the number of prefix normal words of length n. Our lower bound on $pnw(n)$ is proved in Section 6.

Theorem 1. *There exists $c > 0$ such that*

$$pnw(n) = \Omega\left(2^{n - c\sqrt{n \ln n}}\right) = \Omega\left((2 - \varepsilon)^n\right) \qquad \text{for all } \varepsilon > 0. \tag{1}$$

If we consider the length of the first 1-run, we obtain an upper bound.

Theorem 2. *For $n \geq 1$, we have $pnw(n) = O\left(\frac{2^n (\ln n)^2}{n}\right) = o(2^n)$.*

Proof. Let $k = k(n) > 0$ be a number to be specified later. Partition $\mathcal{L}_{\mathrm{PN}} \cap \Sigma^n \setminus \{0^n\}$ into two classes according to the length of the first 1-run.
Case 1: If w is prefix normal and the first 1-run's length is less than k, then there are no k consecutive 1s in w. Write w as the concatenation of $\lfloor n/k \rfloor$ blocks of length k and a final, possibly shorter block: $w = (w_1 \dots w_k)(w_{k+1} w_{k+2} \dots w_{2k}) \dots$ For each block we have at most $2^k - 1$ possibilities, so there can be at most $(2^k - 1)^{\lceil n/k \rceil}$ words in this class. *Case 2:* The length of the first 1-run in w is at least k. Since the first k symbols of w are already fixed as 1s, there can only be $2^{n-k} = 2^n/2^k$ words in this class.

If we balance the two cases by letting k be the largest integer such that $2^k \cdot k^2 \cdot \ln 2 \leq n$, then we have $k = \Theta(\ln n)$ and

$$pnw(n)/2^n \leq \left(1 - \frac{1}{2^k}\right)^{\lceil n/k \rceil} + \frac{1}{2^k} = \Theta\left(\frac{k^2}{n}\right) = \Theta\left(\frac{(\ln n)^2}{n}\right) = o(1),$$

as stated. □

4 Exact formulas for special classes of prefix normal words

Words with fixed density. We formulate an equivalent definition of the prefix normal property that will be useful in the enumeration of prefix normal words. Let $w = 1 w_2 w_3 \dots w_n$ be a prefix normal word of density $d > 0$. Denote by $r_1, r_2, \dots, r_{d-1}$ the distances between consecutive occurrences of 1 in w, and set r_d so that $\sum r_j = n$ holds. We can thus write $w = 10^{r_1 - 1} 10^{r_2 - 1} \dots 10^{r_d - 1}$. For $w = 110100010$, we have $d = 4$, $r_1 = 1$, $r_2 = 2$, $r_3 = 4$ and $r_4 = 2$. The prefix normal property is equivalent to requiring that for all k, one of the shortest substrings containing exactly k ones is a prefix. This gives us the following lemma.

Lemma 2. *The binary word w is prefix normal if and only if the following inequalities hold:*

$$\begin{aligned}
r_1 &\leq r_j && j = 2, 3, \dots, d-3, d-2, d-1 \\
r_1 + r_2 &\leq r_j + r_{j+1} && j = 2, 3, \dots, d-3, d-2 \\
r_1 + r_2 + r_3 &\leq r_j + r_{j+1} + r_{j+2} && j = 2, 3, \dots, d-3 \\
&\vdots && \vdots \\
r_1 + r_2 + \dots + r_{d-2} &\leq r_j + r_{j+1} + \dots + r_{d-1} && j = 2
\end{aligned}$$

Lemma 3. *For $d = 0, \dots, 6$, we have the generating functions $f_d(x) = \sum_{n=1}^{\infty} pnw(n, d) x^n$:*

$$\begin{aligned}
f_0(x) &= \frac{1}{1-x} & f_4(x) &= \frac{x^4}{(1-x^3)(1-x)^3} \\
f_1(x) &= \frac{x}{1-x} & f_5(x) &= \frac{x^5(1+x+x^2)}{(1-x^4)(1-x^2)^2(1-x)^2} \\
f_2(x) &= \frac{x^2}{(1-x)^2} & f_6(x) &= \frac{x^6(1+x+x^2+x^3)}{(1-x^5)(1-x^3)(1-x^2)(1-x)^3} \\
f_3(x) &= \frac{x^3}{(1-x^2)(1-x)^2}
\end{aligned}$$

Similar formulas can be derived for $pnw(n, n-d)$ for small values of d. Unfortunately, no clear pattern is visible for $f_d(x)$ that we could use for calculating $pnw(n)$.

Words with a fixed prefix. We now fix a prefix w and give enumeration results on prefix normal words with prefix w. Our first result indicates that we have to consider each w separately.

Definition 1. *If w is a binary word, let $\mathcal{L}_{ext}(w) = \{w' : ww' \text{ is prefix normal }\}$, and $\mathcal{L}_{ext}(w,m) = \mathcal{L}_{ext}(w) \cap \Sigma^{|w|+m}$. Let $ext(w,m,d) = |\{w' : ww' \text{ is prefix normal of length } |w|+m \text{ and density } d\}|$, and $ext(w,m) = |\mathcal{L}_{ext}(w,m)|$.*

Lemma 4. *Let $v, w \in 1\{0,1\}^*$ be both prefix normal. If $v \neq w$ then $\mathcal{L}_{ext}(v) \neq \mathcal{L}_{ext}(w)$.*

We were unable to prove that the growth of these two extension languages also differ.

Conjecture 1. Let $v, w \in 1\{0,1\}^*$ be both prefix normal. If $v \neq w$ then the infinite sequences $(ext(v,m))_{m\geq 1}$ and $(ext(w,m))_{m\geq 1}$ are different.

The values $ext(w,m,d)$ seem hard to analyze. We give exact formulas for a few special cases of interest. Using Lemma 2, it is possible to give formulas similar to those in Lemma 3 for $ext(w,m,d)$ for fixed w and d. We only mention one such result.

Lemma 5. *For $1 \leq d \leq n$ we have $ext(10, n+d-3, d) = pnw(n,d)$.*

Proof. Let w be an arbitrary prefix normal word of length n and density d with 1 as its first symbol. Insert a 0 before each subsequent occurrence of 1. It is easy to see that this operation creates a bijection between the two sets that we want to enumerate. □

The following lemma lists exact values for $ext(w,|w|)$ for some infinite families of words w.

Lemma 6. *Let $F(n)$ denote the nth Fibonacci number: $F(1) = F(2) = 1$ and $F(n+2) = F(n+1) + F(n)$. Then for all values of n where the exponents are nonnegative, we have the following formulas:*

$$ext(0^n, n) = 1$$
$$ext(1^n, n) = 2^n$$
$$ext(1^{n-1}0, n) = 2^n - 1$$
$$ext(1^{n-2}01, n) = 2^n - 5$$
$$ext(1^{n-2}00, n) = 2^n - (n+1)$$

$$ext((10)^{\frac{n}{2}}, n) = F(n+2) \text{ if } n \text{ is even}$$
$$ext((10)^{\lfloor \frac{n}{2} \rfloor}1, n) = F(n+1) \text{ if } n \text{ is odd}$$
$$ext(10^{n-2}1, n) = 3$$
$$ext(10^{n-1}, n) = n+1$$

Proof. For $w = 1^n$, $w = 1^{n-1}0$, $w = 1^{n-2}01$ and $w = 1^{n-2}00$, it is easy to count those extensions that fail to give prefix normal words. Similarly, for $w = 10^{n-2}1$, $w = 10^{n-1}$ and $w = 0^n$, counting the extensions that give prefix normal words gives the results in a straightforward way.

Let n be even. For $w = (10)^{\frac{n}{2}}$, note that ww' is prefix normal if and only if w' avoids 11. The number of such words is known to equal $F(n+2)$. For n odd, the argument is similar. □

5 Experimental results about prefix normal words

We consider extensions of prefix normal words by a single symbol to the right. It turns out that this question has implications for the enumeration of prefix normal words.

Definition 2. *We call a prefix normal word w extension-critical if $w1$ is not prefix normal. Let $crit(n)$ denote the number of extension-critical words in $\mathcal{L}_{PN} \cap \Sigma^n$.*

Lemma 7. *For $n \leq 1$ we have*

$$pnw(n) = 2pnw(n-1) - crit(n-1) = pnw(n-1)\left(2 - \frac{crit(n-1)}{pnw(n-1)}\right). \tag{2}$$

From this it follows that

$$pnw(n) = 2\prod_{i=1}^{n-1}\left(2 - \frac{crit(i)}{pnw(i)}\right). \tag{3}$$

From Theorem 1 we have:

Lemma 8. *For n going to infinity,* $\liminf crit(n)/pnw(n) = 0$.

We conjecture that in fact the ratio of extension-critical words converges to 0. We study the behavior of $crit(n)/p(n)$ for $n \leq 49$. The left plot in Fig. 2 shows the ratio of extension-critical words for $n \leq 49$. These data support the conjecture that the ratio tends to 0. Interestingly, the values decrease monotonically for both odd and even values, but we have $crit(n+1)/pnw(n+1) > crit(n)/pnw(n)$ for even n. We were unable to find an explanation for this.

The right plot in Fig. 2 shows the ratio of extension-critical words multiplied by $n/\ln n$. Apart from a few initial data points, the values for even n increase monotonically and the values for odd n decrease monotonically, and the values for odd n stay above those for even n.

Conjecture 2. Based on empirical evidence, we conjecture the following:

$$crit(n) = pnw(n)\Theta(\ln n/n), \tag{4}$$

$$pnw(n) = 2^{n-\Theta((\ln n)^2)}. \tag{5}$$

Note that the second estimate follows from the first one by (3).

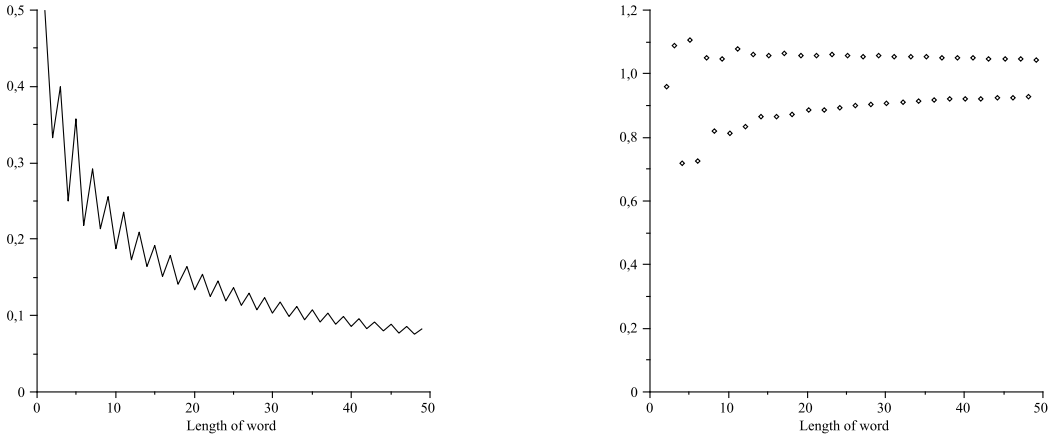

Fig. 2. The ratio $\frac{crit(n)}{pnw(n)}$ (left), and the value $\frac{crit(n)}{pnw(n)} \cdot \frac{n}{\ln n}$ (right).

6 Prefix Normal Games

Variant 1: Prefix normal game starting from empty positions. See Introduction.

Lemma 9. *For $n \geq 7$ Bob has a winning strategy in the game starting from empty positions.*

Variant 2: Prefix normal game with blocks. The game is played as follows. Now a block length of $2k$ is also specified, and we require that $2k$ divides n. The first $4k$ symbols are set to 1 before the game starts (in order to give Alice a fair chance). Divide the remaining empty positions into blocks of length $2k$. Then Bob starts by picking a block with empty positions, and setting half of the positions of the block arbitrarily. Alice moves next and she sets the remaining k positions in the same block as she wants. Now this block is completely filled. Then Bob picks another block, fills in half of it, etc. Iterate this process until every position is filled in.

Lemma 10. *Alice has a winning strategy in the game with blocks, for any $k \geq 1$.*

Proof. Alice can always achieve that the current block contains exactly k 1s and k 0s. Now consider a substring v of length m of the word $w = 1^{4k}u$ that is obtained in the end. We have to show that the prefix of the same length has at least as many 1s. Clearly, only $m \geq 4k$ has to be considered, and we can also assume that v starts after position $4k$. The substring v contains some $2k$-blocks in full, and some others partially. Let $p := \lfloor \frac{m}{2k} \rfloor$, then $|v|_1 \leq (p+1)k \leq \frac{m}{2} + k$, while the number of 1s in the prefix of length m is at least $4k + (p-2)k \geq \frac{m}{2} + k$, as claimed. □

As a corollary, we can prove the lower bound in Theorem 1.

Proof. (of Theorem 1). There are at least as many prefix normal words of length n as there are distinct words resulting after a game with blocks that Alice has won using the above strategy. Note that with this strategy, each block has exactly k many 0s and Bob is free to choose their positions within the block. Moreover, for different choices of 0-positions by Bob, the resulting words will be different. So overall, Bob can achieve at least $\binom{2k}{k}^{(n-4k)/2k}$ different outcomes. If we set $k = \lfloor \sqrt{n \log n} \rfloor$, and note that for $2k$ not dividing n, we can use $pnw(n) \geq pnw(\lfloor n/2k \rfloor \cdot 2k)$, then we obtain: $-\ln(pnw(n)/2^n) = O(\sqrt{n \ln n})$, and the statement follows. □

7 Construction and testing algorithms

In this section, for strings $w \neq 1^n$, we use the notation $w = 1^s 0^t \gamma$, with $s \geq 0, t > 0$ and $\gamma \in 1\Sigma^* \cup \{\varepsilon\}$. Note that this notation is unique. We call $1^s 0^t$ the *critical prefix* of w.

7.1 A mechanical algorithm for computing the prefix normal forms

We now present a *mechanical* algorithm for computing the prefix normal form of a word w. It uses a new algorithm technique we refer to as *sandy beach technique*, a technique that we think will be useful for many other similar problems.

First observe that if you draw your word w as in Fig. 1, then the Parikh set of w will be the region spanned by drawing all the suffixes of w starting from the origin. As we know, the prefix normal forms of w will be the upper and the lower contour of the Parikh set, respectively. This leads to the following algorithm, that we can implement in any sand beach—for example, Lipari's Canneto (Fig. 3).

Take a folding ruler (see Fig. 3) and fold it in the form of your word. Now designate an origin in the sand. Put the folding ruler in the sand so that its beginning coincides with the origin. Next, move it backwards in the sand such that the position at the beginning of the $(n-1)$-length suffix coincides with the origin; then with the next shorter suffix and so on, until the right end of the folding ruler reaches the origin. The traced area to the right of the origin is the Parikh set of w, and its top and bottom boundaries, the prefix normal forms of w (that you can save by taking a photo).

Analysis: The algorithm requires a quadratic amount of sand, but can outperform existing ones in running time if implemented by a very fast person.

Fig. 3. The folding ruler used and a sandy beach (here the beautiful Liparis's Canneto black sand beach) in our mechanical prefix normal construction algorithm.

7.2 Testing algorithm

It can be tested easily in $O(n^2)$ time if a word is prefix normal, by computing its F-function and comparing it to its prefixes; several other quadratic time tests were presented in [14]. Currently, the fastest algorithms for computing F run in worst-case $O(n^2/\text{polylog } n)$ time (references in the Introduction). Here we present another algorithm, which, although $O(n^2)$ in the worst-case, we believe could well outperform other algorithms when iterated on prefixes of increasing length.

Given a word w of length n and density d, $w = 1^s0^t\gamma$. Since the cases $d = 0, n$ are trivial, we assume $0 < d < n$. Notice that, then, in order for w to be prefix normal, $s > 0$ must hold. Now build a sequence of words $v_0, v_1, \ldots, v_{d-s}$, where $v_0 = 1^d0^{n-d}$ and $v_{d-s} = w$, in the following way: for every i, v_{i+1} is obtained from v_i by swapping the positions $d-i$ and j, where j is the rightmost mismatch between v_i and w. So for example, if $w = 110100101$, we have the following sequence of words: 111110000, 111100001, 111000101, 110100101.

The following lemma follows straightforwardly from the results of [8]:

Lemma 11. *Given $w \in \Sigma^n$ with $|w|_1 = d$, and the sequence $v_0 = 1^d0^{n-d}, v_1, \ldots, v_{d-s} = w$, we have that w is prefix normal if and only if every v_i is.*

Moreover, as was shown there, it can be checked efficiently whether these strings are prefix normal. We summarize in the following lemma, and give a proof sketch and an example.

Lemma 12 (from [8]). *Given a prefix normal word $w = 1^s0^t\gamma$. Let $w' = 1^{s-1}0^i10^{t-i}\gamma$, then it can be decided in linear time whether w' is prefix normal.*

We will give an intuition via a picture, see Fig. 4. If w' is not prefix normal, then there must be a k and a substring u of length k s.t. u has more 1s than the prefix of length k. It can be shown that it suffices to check this for one value of k only, namely for $k = s - 1 + t$, the length of the critical prefix length of w'. The number of 1s in this prefix is $s - 1$. Now if such a u exists, then it is either a substring of γ, in which case $F(\gamma, k) > s - 1$; or it is a substring which contains the position of the newly swapped 1 (both in grey in the third line). This latter case can be checked by computing the number of 1s in the prefix of the appropriate length of γ (in slightly darker grey) and checking whether it is greater than $s - 2$.

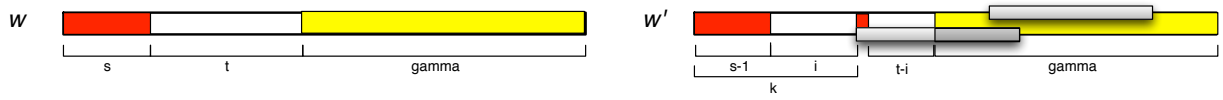

Fig. 4. Proof of Lemma 12.

Thus, for $i = 1, \ldots, d-s$, we test if v_{i+1} is prefix normal. If at some point, we receive a negative answer, then the test returns NO, otherwise it returns YES. Additional data structures for the algorithm are the F-function, which is updated to the current suffix following the critical prefix, up to the length of the next critical prefix (in linear time); and a variable z containing the number of 1s in the appropriate length prefix of γ.

Example: We test whether the word $w = 110101101100100$ is prefix normal.

w	110101101100100	γ	k	$F(k)$	z	F
v_1	111111110000000	ε	12	0	0	000000000000
v_2	111111100000100	100	9	1	1	111111111
v_3	111111000100100	100100	8	2	2	11122222
v_4	111110001100100	1100100	6	3	2	122233
v_5	111100101100100	101100100	5	3	3	12233

At this point we have $z + 1 = 4 > 3 = s - 1$ and therefore, we stop. Indeed, we can see that the next word to be generated, $v_6 = 1110001101100100$ is not be prefix normal, since it has a substring of length 5 with 4 ones, but the prefix of length 5 has only 3 ones.

Analysis: The running time of the algorithm is $O(\sum_{i=d-s}^{d} p_i)$ in the worst case, where the p_i are the positions of the 1s in w, so in the worst case quadratic.

Iterating version. The algorithm tests a condition on the suffixes starting at the 1s, in increasing order of length, and compares them to a prefix where the remaining 1s but one are in a block at the beginning. This implies that for some w which are not prefix normal, e.g. $w = 101^n, n > 1$, the algorithm will stop very late, even though it is easy to see that the word is not prefix normal. This problem can be eliminated by running some linear time checks on the word first; the power of this approach will be demonstrated in the next section.

Since we know that a word w is prefix normal iff every prefix of w is, we have that a word which is *not* prefix normal has a shortest non-prefix-normal prefix. We therefore adapt the algorithm in order to test the prefix normality on the prefixes of w of length powers of 2, in increasing order. In the worst case, we apply the algorithm $\log n$ times. Since the test on the prefix of length 2^i takes $O(2^{2i})$ time, we have an overall $\sum_{i=0}^{\log n} O(2^{2i}) = O(n^2)$ worst case running time, so no worse than the original algorithm.

n	10	12	14	16	18	20	22	24
(a)	2.500	2.561	2.602	2.631	2.656	2.675	2.693	2.708
(b)	2.168	2.142	2.121	1.106	2.093	2.083	2.075	2.067

Table 1. (a) Ratios from the trivial rejection test. (b) Ratios by adding secondary rejection test.

We believe that our algorithm will perform well on strings which are "close to prefix normal" in the sense that they have long prefix normal prefixes, or they have passed the filters, i.e. that it will be expected strongly subquadratic, or even linear, time even on these strings.

7.3 Membership testing with linear time filters

In this section, we provide a two-phase membership tester for prefix normal words. Experimental evidence indicates that on *average* its running time is $O(n)$.

Suppose there is an $O(n)$ test that can be used to reject $2^n - 2^n/n$ of the binary strings outright (Phase I). For the remaining $2^n/n$ strings, apply the worst case $O(n^2)$ algorithm (Phase II). This gives an $O(n)$-amortized time algorithm when taken over all 2^n strings. For such a two-phase approach, let M denote the strings not rejected by the first phase. We are interested in the ratio $nM/2^n$. As n grows, if it appears as though this ratio is bounded by a constant, then we would conjecture that such a membership tester runs in $O(n)$ average case time.

First we try a trivial $O(n)$ test: a string will *not* be prefix-normal if the longest substring of 1s is not at the prefix. Applying this test as the first phase, the resulting ratios for some increasing values of n are given in Table 7.3(a). Since the ratios are increasing as n increases, we require a more advanced rejection test.

The next attempt uses a more compact *run-length* representation for w. Let w be represented by a series of c blocks, which are maximal substrings of the form 1^*0^*. Each block B_i is composed of two integers (s_i, t_i) representing the number of 1s and 0s respectively. For example, the string 11100101011100110 can be represented by $B_1B_2B_3B_4B_5 = (3,2)(1,1)(1,1)(3,2)(2,1)$. Such a representation can easily be found in $O(n)$ time. A word w will *not* be prefix normal word if it contains a substring of the form $1^i0^j1^k$ such that $i+j+k \leq s_1+t_1$ and $i+k > s_1$ (the substring is no longer, yet has more 1s than the critical prefix). Thus, a word will not be prefix normal, if for some $2 \leq i \leq c$:

$$s_{i-1} + t_{i-1} + s_i \leq s_1 + t_1 \quad \text{and} \quad s_{i-1} + s_i > s_1.$$

By applying this additional test in our first phase, we obtain algorithm MemberPN(w), consisting of the two rejection tests, followed by any simple quadratic time algorithm.

The ratios that result from this algorithm are given in Table 7.3(b). Since the ratios are decreasing as n increases, we make the following conjecture.

Conjecture 3. The membership tester MemberPN(w) for prefix normal words funs in average case $O(n)$-time.

We note that there are several other trivial *rejection* tests that run in $O(n)$ time, however these two were sufficient to obtain our desired experimental results.

Acknowledgements. We thank Ferdinando Cicalese who pointed us to [6] and thus contributed to the *fun* part of our paper.

References

1. A. Amir, A. Apostolico, G. M. Landau, and G. Satta. Efficient text fingerprinting via Parikh mapping. *J. Discrete Algorithms*, 1(5-6):409–421, 2003.
2. G. Badkobeh, G. Fici, S. Kroon, and Zs. Lipták. Binary jumbled string matching for highly run-length compressible texts. *Inf. Process. Lett.*, 113(17):604–608, 2013.
3. G. Benson. Composition alignment. In *Proc. of the 3rd International Workshop on Algorithms in Bioinformatics (WABI'03)*, pages 447–461, 2003.
4. S. Böcker. Simulating multiplexed SNP discovery rates using base-specific cleavage and mass spectrometry. *Bioinformatics*, 23(2):5–12, 2007.
5. S. Böcker, K. Jahn, J. Mixtacki, and J. Stoye. Computation of median gene clusters. In *Proc. of the Twelfth Annual International Conference on Computational Molecular Biology (RECOMB 2008)*, pages 331–345, 2008. LNBI 4955.
6. M. Brooks and G. Wilder. Young Frankenstein. http://www.imdb.com/title/tt0072431/quotes, http://www.youtube.com/watch?v=yH97lImrr0Q, 1974.
7. P. Burcsi, F. Cicalese, G. Fici, and Zs. Lipták. On Table Arrangements, Scrabble Freaks, and Jumbled Pattern Matching. In *Proc. of the 5th International Conference on Fun with Algorithms (FUN 2010)*, volume 6099 of *LNCS*, pages 89–101, 2010.
8. P. Burcsi, G. Fici, Zs. Lipták, F. Ruskey, and J. Sawada. On combinatorial generation of prefix normal words. In *Proc. 25th Ann. Symp. on Comb. Pattern Matching (CPM 2014)*, volume 8486 of *LNCS*, pages 60–69, 2014.
9. A. Butman, R. Eres, and G. M. Landau. Scaled and permuted string matching. *Inf. Process. Lett.*, 92(6):293–297, 2004.
10. F. Cicalese, G. Fici, and Zs. Lipták. Searching for jumbled patterns in strings. In *Proc. of the Prague Stringology Conference 2009 (PSC 2009)*, pages 105–117. Czech Technical University in Prague, 2009.
11. F. Cicalese, T. Gagie, E. Giaquinta, E. S. Laber, Zs. Lipták, R. Rizzi, and A. I. Tomescu. Indexes for jumbled pattern matching in strings, trees and graphs. In *Proc. of the 20th String Processing and Information Retrieval Symposium (SPIRE 2013)*, volume 8214 of *LNCS*, pages 56–63, 2013.
12. F. Cicalese, E. S. Laber, O. Weimann, and R. Yuster. Near linear time construction of an approximate index for all maximum consecutive sub-sums of a sequence. In *Proc. 23rd Annual Symposium on Combinatorial Pattern Matching (CPM 2012)*, volume 7354 of *LNCS*, pages 149–158, 2012.
13. K. Dührkop, M. Ludwig, M. Meusel, and S. Böcker. Faster mass decomposition. In *WABI*, pages 45–58, 2013.
14. G. Fici and Zs. Lipták. On prefix normal words. In *Proc. of the 15th Intern. Conf. on Developments in Language Theory (DLT 2011)*, volume 6795 of *LNCS*, pages 228–238. Springer, 2011.
15. T. Gagie, D. Hermelin, G. M. Landau, and O. Weimann. Binary jumbled pattern matching on trees and tree-like structures. In *Proc. of the 21st Annual European Symposium on Algorithm (ESA 2013)*, pages 517–528, 2013.
16. E. Giaquinta and Sz. Grabowski. New algorithms for binary jumbled pattern matching. *Inf. Process. Lett.*, 113(14-16):538–542, 2013.
17. D. Hermelin, G. M. Landau, Y. Rabinovich, and O. Weimann. Binary jumbled pattern matching via all-pairs shortest paths. Arxiv: 1401.2065v3, 2014.
18. T. Kociumaka, J. Radoszewski, and W. Rytter. Efficient indexes for jumbled pattern matching with constant-sized alphabet. In *Proc. of the 21st Annual European Symposium on Algorithm (ESA 2013)*, pages 625–636, 2013.
19. L.-K. Lee, M. Lewenstein, and Q. Zhang. Parikh matching in the streaming model. In *Proc. of 19th International Symposium on String Processing and Information Retrieval, SPIRE 2012*, volume 7608 of *Lecture Notes in Computer Science*, pages 336–341. Springer, 2012.
20. T. M. Moosa and M. S. Rahman. Indexing permutations for binary strings. *Inf. Process. Lett.*, 110:795–798, 2010.
21. T. M. Moosa and M. S. Rahman. Sub-quadratic time and linear space data structures for permutation matching in binary strings. *J. Discrete Algorithms*, 10:5–9, 2012.
22. J. I. Munro. Tables. In *Proc. of Foundations of Software Technology and Theoretical Computer Science (FSTTCS'96)*, pages 37–42, 1996.
23. L. Parida. Gapped permutation patterns for comparative genomics. In *Proc. of the 6th International Workshop on Algorithms in Bioinformatics, (WABI 2006)*, pages 376–387, 2006.
24. F. Ruskey, J. Sawada, and A. Williams. Binary bubble languages and cool-lex order. *J. Comb. Theory, Ser. A*, 119(1):155–169, 2012.
25. N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. Available electronically at `http://oeis.org`. Sequence A194850.
26. A. M. Williams. *Shift Gray Codes*. PhD thesis, University of Victoria, Canada, 2009.