STRING RECOGNITION ON ANONYMOUS RINGS

Evangelos Kranakis [*] [†]
(kranakis@scs.carleton.ca)

Danny Krizanc [*] [†]
(krizanc@scs.carleton.ca)

Flaminia L. Luccio[*] [†]
(luccio@scs.carleton.ca)

Abstract

We consider the problem of recognizing whether a given binary string of length n is equal (up to rotation) to the input of an anonymous oriented ring of n processors. Previous algorithms for this problem have been "global" and do not take into account "local" patterns occurring in the string. Such patterns may be repetitive or discriminating, and can be used to provide efficient algorithms for recognizing strings. In this paper we give new upper and lower bounds on the bit complexity of string recognition. For the case of periodic strings, near optimal bounds are given which depend on the period of the string. For the case of a randomly chosen string, an optimal algorithm for the problem is given. In particular, we show that almost all strings can be recognized by communicating $\Theta(n \log n)$ bits. It is interesting to note that Kolmogorov complexity theory is used in the proof of our *upper* bound, rather than its traditional application to the proof of lower bounds.

1980 Mathematics Subject Classification: 68Q99

CR Categories: C.2.1

Key Words and Phrases: Anonymous rings, boolean function, period, string, Kolmogorov random string.

Carleton University, School of Computer Science: SCS-TR-256

[*]Research supported in part by National Sciences Engineering Research Council of Canada grant.

[†]Carleton University, School of Computer Science, Ottawa, ON, Canada. K1S 5B6

1 Introduction

There have been several studies in the literature concerning the construction of communication efficient algorithms for computing functions on anonymous rings [1, 3, 4, 5, 9, 10], as well as on more general anonymous networks, like tori [2], hypercubes [6], Cayley networks [7], etc. In general, studies of the bit complexity of computing boolean functions of the inputs mainly resort to input collection before determining the output of the boolean function.

In particular, Attiya, Snir and Warmuth [1] show that all the functions that are computable in an anonymous asynchronous ring of processors, can be computed using $O(n^2)$ messages (one bit message if the functions are boolean). They also proved that every algorithm that computes the minimum of all inputs (the OR function in the boolean case), requires as $\Omega(n^2)$ messages, i.e., the bounds in this case are tight. Furthermore they show that almost all boolean functions on n variables, require $\Omega(n^2)$ bits to compute. On the other hand, Moran and Warmuth [10] give an $\Omega(n \log n)$ lower bound for computing any non-constant function and present functions (derived from a recursive application of de Bruijn sequences) with a $O(n \log n)$ bit complexity.

In this paper we further investigate the bit complexity of computing boolean functions on an anonymous ring by concentrating on an interesting class of functions related to the problem of recognizing an input as being equivalent up to rotation to a fixed string. These functions are shown to exhibit behavior opposite to what was described above, i.e., almost all such functions can be computed using $O(n \log n)$ bits and functions requiring $\Omega(n^2)$ bits are the exception, not the rule. The bit complexity of string recognition is shown to be closely related to the underlying symmetry of the string.

1.1 Results of the paper

Let $x = x_1 x_2 \cdots x_n, y = y_1 y_2 \cdots y_n \in \{0,1\}^n$ be binary strings of length n and suppose that for each $i = 1, \ldots, n$ the ith processor of an anonymous ring knows the ith bit y_i of y as well as the entire string x. By string recognition we understand the following problem: "Give an efficient algorithm so that all n processors recognize whether or not the input string y is identical (up to rotation) to the string x".

We can reformulate our problem in terms of boolean functions. For

any string $x \in \{0,1\}^n$ define the boolean function $f_x : \{0,1\}^n \to \{0,1\}$ as follows:

$$f_x(y) = \begin{cases} 1 & \text{if } y \text{ is a cyclic shift of } x \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, being invariant under rotations, the function f_x is computable in the oriented anonymous ring [1]. We are interested in determining the bit complexity of computing $f_x(y)$.

The input collection algorithm of Attiya, Snir and Warmuth [1] implies that for every x, $f_x(y)$ can be computed using $O(n^2)$ bits. Such an algorithm is "global", in the sense that the same algorithm is executed regardless of x or the input to the network. In the sequel, we show how taking into account repetitive or discriminating patterns within a string might provide more efficient algorithms for our problem. For the case of symmetric or repetitive strings we specify upper bounds (section 2.1) and lower bounds (section 3) which depend on the period k of a string x of length n, where $x = w^t v$, $t \geq 0$, v a prefix of w, and $k = |w|$. If $v = \emptyset$ then we give an $O(nk + n^2 \min\{1, \frac{\log n}{k}\})$ upper bound and $\Omega(\frac{n^2}{k} + n \log n)$ lower bound. If $v \neq \emptyset$ then we give an $O(nk + n \log n)$ upper bound and $\Omega(n \log n)$ lower bound.

Finally, we show (section 2.2) that almost all strings can be recognized using $\Theta(n \log n)$ bits. Our upper bound is an interesting application of Kolmogorov complexity theory which has traditionally be used to show lower bounds. From an intuitive point of view, Kolmogorov complexity theory deals with the amount of information that a string contains [8, 11]. This quantity can be calculated in terms of the size of the shortest program that computes the string. For a random string x of length n, the length of its shortest program is very close to n, with high probability. A string with this property is called incompressible. We use this fact to derive an upper bound of $O(n \log n)$ bits for almost all strings, by showing that if no efficient algorithm exists to recognize a string then a short program exists to compute the string, contradicting its incompressibility.

The following table summarizes the bit complexity of recognizing a string x which is of the form $w^t v$, where $|w| = k$, k is the period of x and v is a proper prefix of w, and for a string x which is Kolmogorov random (see definition below).

$x = w^t v$, $v = \emptyset$	$x = w^t v$, $v \neq \emptyset$	x is K-random
$O(nk + n^2 \min\{1, \frac{\log n}{k}\})$	$O(nk + n \log n)$	$O(n \log n)$
$\Omega(\frac{n^2}{k} + n \log n)$	$\Omega(n \log n)$	$\Omega(n \log n)$

1.2 Preliminaries

As our model of computation we consider the standard anonymous, asynchronous, ring of n processors [1]. The size n of the ring is known to the processors, and initially the processors are given a single bit as well as a string x of length n. By anonymity, the processors execute the same algorithm given the same data. At the end, it is desired that all the processors determine correctly whether or not the given string is a rotation of the input.

In the proofs below we further assume that the ring is oriented, i.e., the processors can distinguish in a "globally consistent" manner their left from their right neighbor. Nevertheless, all our results are valid for unoriented rings with only minor modifications in the definitions and the statements of the results.

Let n be the number of processors and let $0 \leq i < n$ denote the ith processor. If b_p is the input to processor p then $< b_p : p < n >$ denotes the network input. The right (respectively, left) neighborhood of processor p of length k is the sequence of bits $< b_{p+i} : 0 \leq i < k >$ (respectively, $< b_{p-i} : 0 \leq i < k >$), where the operations $+, -$ are modulo n.

Before we proceed with the main algorithm we give some necessary definitions.

NOTATION **1** *Call u (respectively, v) a* prefix *(respectively,* suffix*) of the string x, if $x = uv$ for some string v (respectively, u); the prefix (respectively, suffix) is called* proper *if it is not equal to x. For any string x of length n let the* period *of x be the smallest integer $k \leq n$ such that for some strings w, v we have that*

- *w has length k,*
- *v is a prefix of w, and*
- *$x = w^{\lfloor n/k \rfloor} v$.*

For any string u, let $|u|$ denote its length.

NOTATION **2** *: Given a string $x = x_1 x_2 \ldots x_n$ of length n, let $x \uparrow s = x_1 x_2 \ldots x_s$ denote the first s bits of x, and $x^r = x_{r+1} x_{r+2} \ldots x_n x_1 x_2 \ldots x_r$ the r-th rotation of x. S_x^t is the set of all the different substrings of x, and of all its rotations, that are of length $t \leq n$, i.e., $S_x^t = \{x^i \uparrow t \,|\, i = 1, \ldots, n\}$.*

DEFINITION **3** *: Given a string x of length n, a program p of length $|p|$, a programming language S, let $K_S(x)$ denote the length of the minimum program that outputs x using S, i.e., $K_S(x) = \min\{|p| : S(p) = x\}$.*

From now on we will use the notation $K(x)$ to refer to the Kolmogorov complexity of x with respect to an optimal method of specification S, i.e., a method whose complexity differs by at most a constant from other methods that compute x. If $K(x) \geq |x| - O(\log |x|)$ we say that the string x is loosely *random*, or $O(\log |x|)$-*incompressible*, or *Kolmogorov random*. It is easy to derive the fact that almost every string of a given length is Kolmogorov random. To simplify our presentation we assume from now on that the string x is Kolmogorov random if $K(x) \geq |x| - \log |x|$. In particular, this implies that a string of length n is Kolmogorov random with probability at least $1 - 1/n$.

2 Upper Bounds

2.1 Periodic strings

In the sequel, we give two efficient algorithms for the string recognition problem. We can prove the following theorem.

THEOREM 4 *Let $x = w^{\lfloor n/k \rfloor} v$ be a string of period k such that $|x| = n, |w| = k$ and v is a proper prefix of w. There is an algorithm for recognizing the string x on an n processor anonymous oriented ring with bit complexity $O(nk + n^2 \min\{1, \frac{\log n}{k}\})$ if $v = \emptyset$, and $O(nk + n \log n)$ if $v \neq \emptyset$.*

PROOF The $O(n^2)$ upper bound is an immediate consequence of the standard input collection algorithm given in [1]. So we concentrate on proving the rest of the upper bound.

The main proof of the theorem consists of an efficient input collection algorithm satisfying the requirements above. Let a string x be known to all the processors and let b_p be the input to processor p. In the output of the algorithm it is determined whether or not x is a rotation of the input. Each processor executes the following algorithm which is described in several phases.

Phase 1

The processor computes the period k of the given string x as well as strings w, v such that v is a proper substring of w and in addition $x = w^t v$, for some integer $t \geq 0$. This is a local step executed initially by all the processors and does not contribute in any way to the overall bit complexity of the final algorithm.

Phase 2

Let k be the period of the string x as computed in phase 1. The processors now compute their left neighborhood of length k as well as their right neighborhood of length $2k+1$ by executing the following algorithm.

/* computing the right neighborhood */
send b_p to the left;
for $i = 1$ to $2k$ **do**
    send bit received from the right to the left
    od
/* computing the left neighborhood */
send b_p to the right;
for $i = 1$ to $k-1$ **do**
    send bit received from the left to the right
    od

The idea of this phase is to compute a *small* length neighborhood of the input string and use this to decide whether or not the input string is a rotation of x. It will become apparent in the case $v \neq \emptyset$ (discussed in the sequel) why we need to compute a neighborhood of length $2k+1$ to the right and a neighborhood of length k to the left (for the time being we will only have use for the left neighborhood of length k). By phase 1 each processor knows the representation $x = w^t v$. Clearly, the bit complexity of phase 2 is $O(nk)$.

Phase 3

In this phase the processors will execute the main part of the algorithm that will determine whether or not the given string is a rotation of the input. The rest of the algorithm depends on whether or not $v = \emptyset$.

CASE $v = \emptyset$:

Let u_p be the first k bits of the right neighborhood computed by processor p in phase 2 above. Now p executes the following algorithm.

if u_p is not a rotation of w
  then broadcast x is not a rotation of the input;
  else /* u_p is a rotation of w */

if u_p is the lexicographically maximal rotation of w
  then send two counters c_0, c_1 around the ring;
  counter c_0 is incremented only by processors
  whose string is a rotation of w;
  counter c_1 is incremented by every processor;
if processor p receives counter values such that
  $c_0 = c_1 = n$
  then broadcast x is a rotation of the input;
  else broadcast x is not a rotation of the input;
fi

At the confirmation stage at most n/k processors participate (i.e., the ones that have a lexicographically maximal rotation of w); each processor sends a message around the ring (confirming that it has a lexicographically maximal rotation of w) together with a counter indicating that the message has circulated all the way around the ring (whence the $\log n$ factor). Finally, if any processor either has a neighborhood of length k which is not a rotation of w or receives a message that such a neighborhood exists in the ring, it sends this message to its neighbor and does not transmit again. Thus the overall bit complexity of this phase is $O(n^2 \log n/k)$.

At the end of this phase all processors will know whether or not all processors have seen a rotation of w. If indeed, all processors have seen a rotation of w then the string x is accepted else the string is rejected. The correctness of this last assertion is proved in the sequel.

In the following argument we assume that the strings x, w, v are as in the statement of Theorem 4.

If there exists a processor $p \in \{i = 1, \ldots, n\}$ that does not see a rotation of the input, this obviously implies $x \neq w^t$, and p will broadcast a message to stop the other processors.

If all the processors have seen a rotation of the input, then $x = w^t$ must hold. Let us assume on the contrary that there exist a processor p whose left neighborhood $w^{(t-1)} = w_t \ldots w_k w_1 \ldots w_{t-1}$, $t = 1, \ldots, k$ is a rotation of the input, but its neighbor $p+1$ sees another rotation $w^{(s-1)} = w_s \ldots w_k w_1 \ldots w_{s-1}$, $s = 1, \ldots, n$, which is not adjacent. Since p and $p+1$ are adjacent, their left neighborhood must have $k-2$ bits that are the same, i.e., $w_{(t+1) \bmod k} \ldots w_k w_1 \ldots w_{t-1} = w_s \ldots w_k w_1 \ldots w_{s-2}$. Moreover, if the two rotations cannot be adjacent, we have $w_t \neq w_{s-1}$, and since we are working with binary strings, we have $w_t = \overline{w_{s-1}}$. We now need the following:

LEMMA **5** *Every two rotations* $w^{(t-1)} = w_t \ldots w_k w_1 \ldots w_{t-1}$, $t = 1, \ldots, k$,

and $w^{(s-1)} = w_s \ldots w_k w_1 \ldots w_{s-1}$, $s = 1, \ldots, k$ *of the binary string* $w^{(0)} = w_1 \ldots w_k$ *of length* k *have the same number of ones.*

PROOF Let us assume by contradiction that the number of ones in $w^{(t-1)}$ and $w^{(s-1)}$ is different. By definition of rotation, if we start rotating both the bits of $w^{(t-1)}$ and of $w^{(s-1)}$ at a certain point we obtain the string $w^{(0)}$, and this implies that the number of ones must be the same. ■

We have assumed that there can be two rotations with a different number of zeros since $w_t \neq w_{s-1}$, but the above lemma shows that this is not possible, i.e., we have a contradiction. If each processor sees a rotation of the input, then the string can only be $x = w^t$.

CASE $v \neq \emptyset$:

We assume that $x = w^t v$, where $v = v_1 \ldots v_l$ is a proper prefix of w of length l, i.e., $v = w_1 \ldots w_l$. Using the fact that w is the period of the string x, and assuming $x = x_1 \ldots x_n$, we can prove easily the following lemma.

LEMMA **6** *There exists a substring* $z = z_1 \ldots z_k$ *of length* k *of* x *that has weight (i.e., number of bits equal to* 1*) different from the weight of* w*.*

PROOF Easy by contradiction, using the fact w is the period of x, and that $v \neq \emptyset$. ■

Let v_p be the neighborhood of length $3k$ computed by processor p in phase 2. It consists of the left neighborhood of length $2k + 1$ and the right neighborhood of length k. In view of the representation of x in the form $w^t v$ it is an immediate consequence of Lemma 6 that there is a *unique* position in the string x whose k-neighborhood is not a rotation of w, but such that the k-neighborhoods of all k positions to its left are indeed rotations of w and the k positions to its right have the *correct* values.

However, the converse of this is also true, in the sense that if the input string has a unique position satisfying the above conditions, as well as the k positions to its right have the correct k-neighborhoods, and in addition all other positions have a k neighborhood which is a rotation of w then in fact the input string itself must be a rotation of x. In the proof of the following lemma we assume that $x = w^{\lfloor \frac{n}{k} \rfloor} v$, $\lfloor \frac{n}{k} \rfloor \geq 2$. The case $x = wv$ can be solved using input collection with $O(n^2)$ bit complexity.

LEMMA **7** *If after phase 2 of the algorithm there exists at least a processor whose* $3k$*-bits string obtained in Phase 2 is different from* $3k$ *bits of the*

string w^3vw^3, then $x \neq w^{\lfloor \frac{n}{k} \rfloor}v$. If on the other hand all the processors see a correct $3k$-neighborhood, then $x = w^{\lfloor \frac{n}{k} \rfloor}v$, or $x = w^{s_1}vw^{s_2}v \ldots w^{s_i}v$, for $s_j \geq 2$, $j = 1, \ldots i$, $i \geq 2$.

PROOF The first part of the proof is straightforward. For the second part let us then assume that the $3k$-bits string is correct, and let us examine the $3k$ (correct) bits that two processor that are neighbors can see. We will denote them with $Y_1 = y_1 \ldots y_{3k}$, and $Y_2 = y_2 \ldots y_{3k+1}$. We will now show that the only strings that can occur are $x = w^{\lfloor \frac{n}{k} \rfloor}v$, or $x = w^{s_1}vw^{s_2}v \ldots w^{s_i}v$, for $s_j \geq 2$, $j = 1, \ldots, i$, $i \geq 2$. There are four cases:

1. Y_1 and Y_2 are both rotations of w^3.

   This implies that $y_1 = y_{3k+1}$ (same as the proof for the case $v = 0$).

2. Y_1 is a rotation of w^3 and Y_2 is a rotation of $3k$ bits of w^3vw^3, but $Y_2 \neq w^3$.

   Since $v_1 \ldots v_l = w_1 \ldots w_l$ and Y_1 and Y_2 are consecutive, for $Y_1 = w_s \ldots w_k w^2 w_1 \ldots w_{s-1}$ we have:

   If $s \leq k$, then $Y_2 = w_{s+1} \ldots w_k w^2 w_1 \ldots w_{s-1} y_{3k+1}$ for $s < k$, or $Y_2 = w^2 w_1 \ldots w_{k-1} y_{3k+1}$ for $s = k$. This implies $y_{3k+1} \neq w_s$ because Y_2 is not a rotation of w^3, so it must be $y_{3k+1} = w_1$ and $l = s - 1$. There are then $3k + 1$ consecutive bits in the resulting string, in the form $w_s \ldots w_k w^2 v w_1$, $|v| = s - 1$.

3. Y_1 is a rotation of $3k$ bits of w^3vw^3, but $Y_1 \neq w^3$, and Y_2 is a rotation of w^3.

   Since $v_1 \ldots v_l = w_1 \ldots w_l$ and Y_1 and Y_2 are consecutive, for $Y_2 = w_t \ldots w_k w^2 w_1 \ldots w_{t-1}$ we have:

   (a) If $t \geq 2$, then $Y_1 = y_1 w_t \ldots w_k w^2 w_1 \ldots w_{t-1}$, and $y_1 \neq w_{t-1}$ because Y_1 is not a rotation of w^3. So it must be $y_1 = w_k \neq w_{t-1}$, and $w_t \ldots w_k = w_1 \ldots w_{k-t+1} = v_1 \ldots v_l$, and $k - t + 1 = l$. If this is not true, the two neighbors cannot have seen a correct neighborhood. There are then $3k + 1$ consecutive bits in the form $w_1 w_t \ldots w_k w^2 w_1 \ldots w_{t-1} = w_1 v w^2 w_1 \ldots w_{k-l}$, $|v| = k - t + 1$.

   (b) If $t = 1$, then $Y_1 = y_1 w^2 w_1 \ldots w_{k-1}$, and $y_1 \neq w_k$ because Y_1 is not a rotation of w^3, so it must be $y_1 = v_l \neq w_k$. There are then $3k + 1$ consecutive bits in the form $v_l w^2 w_1 \ldots w_{k-1}$.

4. Both Y_1 and Y_2 are rotations of $3k$ bits of w^3vw^3, but $Y_1, Y_2 \neq w^3$.

This case is much more complicated then the others since the substring v can be in different positions within the $3k$ bits. In most of the cases we will assume that Lemma 6 is true, i.e., by looking at k consecutive bits, we can always tell whether they are a rotation of w or not. We will refer to the following:

LEMMA **8** *The strings* $w = w^{(0)}, w^{(1)}, \ldots, w^{(k-1)}$ *are all different.*

PROOF If on the contrary any two of the above strings were equal then we could find a string u such that $u^r = w$, for some integer $r > 1$, thus contradicting the fact that w is the period of the string x. ■

(a) $Y_1 = w_s \ldots w_k w^2 v_1 \ldots v_l w_1 \ldots w_{s-l-1}$ (i.e., $w_1 \neq w_{l+1}$), $s \leq k, l < s-1$. This implies $Y_2 = w_{s+1} \ldots w_k w^2 v_1 \ldots v_l w_1 \ldots w_{s-l-1} y_{3k+1}$ for $s < k$, or $Y_2 = w^2 v_1 \ldots v_l w_1 \ldots w_{k-l-1} y_{3k+1}$ for $s = k$. The only possible situation is $y_{3k+1} = w_{s-l}$, otherwise the processor don't see a correct configuration. The $3k$ consecutive bits are then $w_s \ldots w_k w^2 v w_1 \ldots w_{s-l}$, $s \leq k$.

(b) i. $Y_1 = w_s \ldots w_{l+1} \ldots w_k w v w_1 \ldots w_{k-l+s-1}$ (i.e., $w_1 \neq w_{l+1}$), $1 \leq s < l+1 \leq k$. The only possible situation is then $Y_2 = w_{s+1} \ldots w_k w v w_1 \ldots w_{k+s-l-1} y_{3k+1}$ and $y_{3k+1} = w_{k-l+s}$. The $3k$ consecutive bits are then $w_s \ldots w_k w v w_1 \ldots w_{k-l+s}$.

ii. $Y_1 = w_{l+1} \ldots w_k w v w_1 \ldots w_{k-l+s-1}$ (i.e., $w_1 \neq w_{l+1}$), $1 \leq s = l+1 < k$. This implies $Y_2 = w_{l+2} \ldots w_k w v w y_{3k+1}$. The only possible situation is $y_{3k+1} = w_1$ (Lemma 8), and the $3k$ consecutive bits are then $w_{l+1} \ldots w_k w v w w_1$.

iii. $Y_1 = w_s \ldots w_k w v w w_1 \ldots w_{s-l-1}$ (i.e., $w_1 \neq w_{l+1}$), $l+1 < s \leq k$. This implies $Y_2 = w_{s+1} \ldots w_k w v w w_1 \ldots w_{s-l-1} y_{3k+1}$ for $s < k$, $Y_2 = w v w w_1 \ldots w_{s-l-1} y_{3k+1}$ for $s = k$. The only possible situation is $y_{3k+1} = w_{s-l}$, otherwise the processors don't see a correct configuration. The $3k$ consecutive bits are then $w_s \ldots w_k w v w w_1 \ldots w_{s-l}$.

(c) i. $Y_1 = w_s \ldots w_{l+1} \ldots w_k v_1 \ldots v_l w w_1 \ldots w_{k-l+s-1}$, $1 \leq s < l+1 \leq k$. This implies $Y_2 = w_{s+1} \ldots w_k v w w_1 \ldots w_{k-l+s-1} y_{3k+1}$. The only possible situation is $y_{3k+1} = w_{k-l+s}$, and the $3k$ consecutive bits are then $w_s \ldots w_k v w w_1 \ldots w_{k-l+s}$.

ii. $Y_1 = w_{l+1} \dots w_k v_1 \dots v_l w^2$, $1 \leq s = l+1 \leq k$. Note that this is different from w^3 for Lemma 8. This implies $Y_2 = w_{l+2} \dots w_k v w^2 y_{3k+1}$. The only possible situation is $y_{3k+1} = w_1$. The $3k$ consecutive bits are then $w_{l+1} \dots w_k v w^2 w_1$.

iii. $Y_1 = w_s \dots w_k v_1 \dots v_l w^2 w_1 \dots w_{s-l-1}$, $l+1 < s < k$. This implies $Y_2 = w_{s+1} \dots w_k v w^2 w_1 \dots w_{s-l-1} y_{3k+1}$ for $s < k$, or $Y_2 = v w^2 w_1 \dots w_{k-l-1} y_{3k+1}$ for $s = k$. I.e., if $(w_{s+1} \dots w_k v = w_{k+l-s+1} \dots w_k)$ for $s < k$ (or $(v = w_{k-l} \dots w_k)$ for $s = k$), and $2l = s - 1$, then $y_{3k+1} = w_1$, else $y_{3k+1} = w_{s-l}$. The $3k$ consecutive bits are then $w_s \dots w_k v w^2 v w_1$ in the first case, or $w_s \dots w_k v w^2 w_1 \dots w_{s-l}$ in the second.

iv. $Y_1 = w_k v_1 \dots v_l w^2 w_1 \dots w_{k-l-1}$, $l+1 < s = k$. This implies $Y_2 = v_1 \dots v_l w^2 w_1 \dots w_{k-l-1} y_{3k+1}$. The only possible situation are if $v_1 \dots v_l = w_{k-l} \dots w_k$ and $k = 2l - 1$ then $y_{3k+1} = w_1$, else $y_{3k+1} = w_{k-l}$. The $3k$ consecutive bits are then $w_k v w^2 v w_1$ in the first case, and $w_k v w^2 w_1 \dots w_{k-l}$ in the second.

v. $Y_1 = v_m \dots v_l w^2 w_1 \dots w_{k+m-l+1}$, $1 \leq m \leq l$. This implies $Y_2 = v_{m+1} \dots v_l w^2 w_1 \dots w_{k+m-l} y_{3k+1}$, i.e., if $v_{m+1} \dots v_k = w_{k+m-l-1} \dots w_k$, and $w_m \neq w_{k-l+m}$, and $2l = k + m$, then $y_{3k+1} = w_1$, else $y_{3k+1} = w_{k+m-l}$. The $3k$ consecutive bits are then $v_m \dots v_l w^2 w_1 \dots w_{k+m-l}$ in the first case, or $v_m \dots v_l w^2 w_1 \dots w_{k+m-l-1} w_1 = v_m \dots v_l w^2 v w_1$ in the second.

It can never happen that two strings v are close to each other less then $2k$ bits, i.e., of a string w^2. This condition comes from the fact that, if we try to build up a string of length n starting from Y_1 and shifting one bit to the right at the time, the resulting string will maintain the above condition. This can be viewed as follows: if a new string $Y_3 = y_3 \dots y_{3k+3}$ is built starting from the string Y_2 we obtained before, we will fall again in one of the above cases. The only critical case is case 4 (c) iii), iv), and v), where the string v occurs twice while building Y_2 from Y_1, but Y_2 sees only the rightmost one. Y_3 will anyway see the string v to the right of w^2, i.e., Y_2 and Y_3 will fall again in case (4) a and so on. This implies that the only possible resulting strings are either $x = w^{\lfloor \frac{n}{k} \rfloor} v$, or $x = w^{s_1} v w^{s_2} v \dots w^{s_i} v$, for $s_j \geq 2$, $j = 1, \dots i$, $i \geq 2$.

■

It is now easy to convert this characterization into an $O(nk + n \log n)$ algorithm for determining whether or not the input is a rotation of x. The

algorithm for processor p is as follows.

if a proper rotation is not seen **or** a halt is received
  then broadcast x is not a rotation of the input and halt
  else if in the "unique" position
    then send a message around the ring with a counter c_1
  fi;
  counter c_1 is incremented by every processor;
  if in the "unique" position,
  and a counter $c_1 < n$ is received,
  then broadcasts x is not a rotation of the input and halt
  else if $c_1 = n$,
    then broadcast x is a rotation of the input;
    fi
  fi
fi

The observations made above prove the correctness of the algorithm. Only one processor can be in the unique position. If there is more then one, this situation is detected by another processor in a unique position, by receiving $c_1 < n$. Note that it is not possible that every processor has w^3 as a neighborhood ($|x| = n$ cannot be divided by $|w|$, since $|v| < |w|$). The bit complexity of the algorithm is straightforward. In the worst case in this step, a message of at most $\log n$ bits travels for n steps. The total bit complexity of the algorithm, in the case $v \neq 0$ is then $O(nk + n \log n)$. This completes the proof of Theorem 4. ■

2.2 Kolmogorov random strings

We now present an algorithm that optimally finds using $\Theta(n \log n)$ bits, whether a Kolmogorov random string x is a rotation of the input checking only a logarithmic number of bits. (The lower bound follows from [10].) In particular this shows that almost all strings can be recognized using $\Theta(n \log n)$ bits.

THEOREM **9** *Let x be a Kolmogorov random string of length n. There is an algorithm for recognizing the string x on an n processor anonymous oriented ring with bit comlexity $O(n \log n)$.*

PROOF Let x be a Kolmogorov random string of length n, given as an input to all the processors, and let b_p be a single bit of input at processor p. In the output of the algorithm it is determined whether or not x is a rotation of the input configuration. In the algorithm we use a constant c that will be more precisely calculated in Lemma 11. We will refer to Lemma 11 and 12 to justify the fact that, if the $c\lceil \log n \rceil$ neighborhood of a processor coincides with one of the strings contained in $S_x^t = \{x^i \uparrow t \mid i = 1, \ldots, n\}$ (see section 1.2), then, for n large enough, the input string must be x or one of its rotations.

Let $c = \max\{c', c''\}$, where c' and c'' are constants given in Lemma 11 and 12.

A generic processor p executes the following phases.

Phase 1

compute and store $S_x^t = \{x^i \uparrow t \mid i = 1, \ldots, n\}$, for $t = c \log n$. This is a local step and it does not contribute on the overall bit complexity of the final algorithm.

Phase 2

compute the right neighborhood of length t.
/*computing the right neighborhood*/
send b_p to the left;
for $i = 1$ to t **do**
send bit received from the right to the left;
od check whether the t-neighborhood coincides with one of the strings contained in $S_x^t = \{x^i \uparrow t \mid i = 1, \ldots, n\}$. The cost of this phase, in terms of bits transmitted, is $O(n \log n)$.

Phase 3

In this phase p will determine if the input string is a rotation of x.

Since x is a Kolmogorov random string, if after $t - 1$ steps all the processors have seen strings of $S_x^t = \{x^i \uparrow t \mid i = 1, \ldots, n\}$, then the input string must be x (see Lemma 11), otherwise, if at least one finds a different substring, it is not x.

Let u_p be the t-neighborhood of processor p. Now p executes the following:

if $u_p \notin S_x^t = \{x^i \uparrow t \mid i = 1, \ldots, n\}$,
then broadcast x is not a rotation of the input and halt;
else if u_p is the maximal string
then send a counter c_0 around the ring;
/* counter c_0 is only incremented by processors who have seen an acceptable string */

if a counter value $c_0 = n$ is received
then broadcast x is a rotation of the
input and halt;
else /* $c_0 < n$, or another processor
has seen a not acceptable string and has sent a
halt */
broadcasts x is not a rotation of the
input and halt
fi
else /* u_p is not a maximal string */
if a not acceptable string is seen
then broadcast x is not a rotation of the
input and halt
else if a counter c_0 is received
then broadcast $c_0 := c_0 + 1$
else /*a halt is received*/
store the information "x is (is not) a
rotation of the input", broadcast it and halt
fi
fi
fi
fi

We can now precisely define a method to find the right value of c and we can justify phase 2 and 3 of the algorithm.

DEFINITION **10** *Two strings $x = x_1x_2 \ldots x_n$ and $y = y_1y_2 \ldots y_n$ of length n, a substring of y^i $s = s_1 \ldots s_t = a_2 \ldots a_t v_1$, of length $t < n$, $1 \leq i \leq n$, and a substring $a_1 \ldots a_t u_1$ of x^j, $1 \leq j \leq n$ are given. We define as* right overlap *(respectively* left overlap*) of length $l \geq 1$ of $a = a_1 \ldots a_t$ in x^j and s, the substring $a_{t-l+1} \ldots a_t$ (respectively $a_1 \ldots a_l$) of a that coincides with the substring $s_1 \ldots s_l$ (respectively $s_{t-l+1} \ldots s_t$) of s. If $l = 0$ we say that there is no overlap.*

We have the following:

LEMMA **11** *: There exist a constant $c' \in \mathcal{N}$, such that, for n sufficiently large and for all $t \geq c' \log n$, for any two strings $x = x_1x_2 \ldots x_n$ and $y = y_1y_2 \ldots y_n$ of length n, if x is Kolmogorov random, and $S_y^t \subseteq S_x^t$, then $x = y$ up to rotation.*

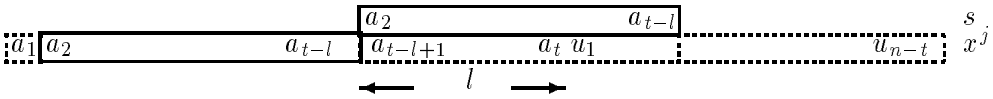

Figure 1: $0 < l < \frac{t}{2}$, $n \geq 1 + 2(t - l - 1)$

PROOF Given an integer $t < n$, let $a \in S_y^t$, i.e., $y^i = a \cdot v = a_1 \dots a_t v_1 \dots v_{n-t}$ for some $i \in \{1, \dots, n\}$ and some string v, $|v| = n - t \geq 1$. Since $a \in S_x^t$, it is clear that $x^j = a \cdot u = a_1 \dots a_t u_1 \dots u_{n-t}$ for some $j \in \{1, \dots, n\}$ and some string u, $|u| = n - t \geq 1$. Consider the substring s of y^i, $s = s_1 \dots s_t = a_2 \dots a_t v_1$ of length t, $s \in S_y^t \subseteq S_x^t$. The proof of the lemma is divided in different cases depending on the different overlaps of the strings s and a in x^j. W.l.o.g we assume that the overlap does not coincide with the bits $a_2 \dots a_t u_1$ since, if it is the case, we can always rotate the string x^j up to a point in which it does not coincide, or otherwise we easily obtain the condition $x = y$ up to rotation.

Let l be the length of the overlap of s in x^j.

- Case 1): There is a right overlap of a in x^j and s, of length $0 < l < \frac{t}{2}$. Moreover $n \geq 1 + 2(t - l - 1)$. (See Figure 1.)

  The overlap covers the bits $a_{t-l+1} \dots a_t$. There might even be a left overlap on a, but w.l.o.g. we will not consider this condition (since, if it is the case, we are able to compress more the coding of the string x). Let us assume $s_1 = a_{t-l+1} = a_2, \dots, s_{l-1} = a_t = a_l, \dots, s_l = u_1 = a_{l+1}, \dots, s_t = u_{t-l} = v_1$. Hence the substring $a_2 \dots a_{t-l}$ of length $t - l - 1 \geq \frac{t}{2}$ is repeated at least twice in x^j.

  We now prove that $K(x) < n - \log n$ which contradicts the incompressibility of x. We encode x as a_1 plus twice a string of length $t - l$, plus the remaining string x' of length $n - 2(t - l) - 1$. Hence x is given first by specifying a_1 in $O(1)$.. The encoding of the two strings $a_2 \dots a_{t-l}$ is as follows: let T be a Turing Machine that computes $a_2 \dots a_{t-l}$ from a program p. Let $num(T) = n$ be the number associated with the Turing Machine T and fix a Universal Turing Machine V such that V started on the input $O^{num(T)}1p$ simulates T on input p and then doubles the output starting from position 2. If we start the reference Universal Turing Machine U on input $O^{num(V)}1O^{num(T)}1p$ we will have the double string as output, starting from position 2. This will add a term

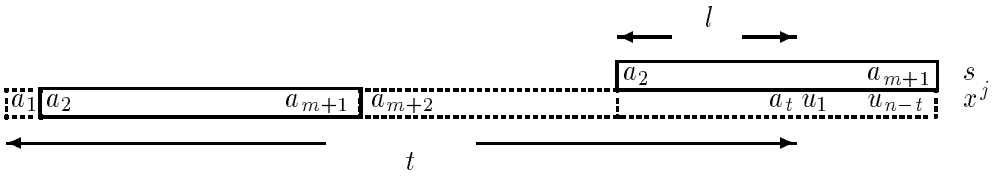

Figure 2: $0 < l < \frac{t}{2}$, $n < 1 + 2(t - l - 1)$

$K(a_2 \ldots a_{t-l}) + O(1)$ for the encoding of twice $a_2 \ldots a_{t-l}$. We have a term of $O(\log 2(t-l) - 1)$ to remember the position of the end of the double string plus $O(\log j)$ bits for the index j of the rotation of the string x^j. The string $x' = u_{t-l+1} \ldots u_{n-t}$ has length $n - 2(t-l) - 1$.

We have then:

$$\begin{aligned} K(x) &\leq O(1) + K(a_2 \ldots a_{t-l}) + O(1) + O(\log(2(t-l)-1)) \\ &+ O(\log j) + K(x') \\ &\leq t - l - 1 + O(\log n) + n - 2(t-l) - 1 \\ &= n - (t-l) + O(\log n) < n - \frac{t}{2} + c_1 \log n, \end{aligned}$$

i.e., there exists a constant $c_1 \in \mathcal{N}$, such that, for large enough n, for $t > 2(c_1 + 1)\log n$, $K(x) < n - \log n$ which is a contradiction.

- Case 2): There is a right overlap of a in x^j and s, of length $0 < l < \frac{t}{2}$. Moreover $n < 1 + 2(t - l - 1)$, i.e., there is even a left overlap. (See Figure 2.)

  The right overlap covers the bits $a_{t-l+1} \ldots a_t$. There are two cases depending on the length of the substring of s that covers the last bits of x^j from position a_{t-l+1}, up to u_{n-t} (since the overlap is wrapping around). To be more precise let $m = l + n - t < t - l - 1$ be the length of this substring. A possible coding is given by repeating twice the last m bits of x^j . We have to remember the positions $m + 1 = l = n - t + 1$ and $t - l + 1$ where respectively the first of these repeated strings ends and the where the other starts. The encoding is given again by running the Turing Machine T on input p' for which it computes the string $a_2 \ldots a_{m+1}$. The Universal Turing Machine V will repeat the string twice, starting from a_2 and a_{t-l+1}. We have then

$$K(n) \leq O(1) + m + K(a_{m+1} \ldots a_{t-l}) + O(\log(m+1))$$

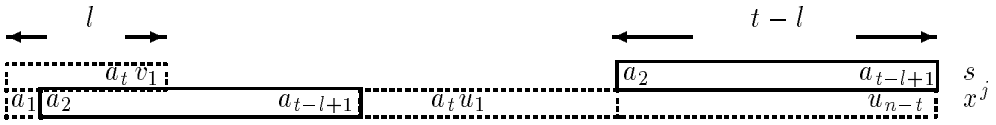

Figure 3: $0 < l < \frac{t}{2}$, $n > 2t - l + 1$

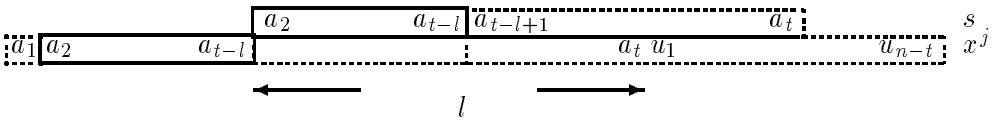

Figure 4: $l \geq \frac{t}{2}$

$$\begin{aligned} & + \quad O(\log t - l + 1) + O(\log j) \\ & = \quad O(1) + m + t - l - m + O(\log n) \\ & = \quad t - l + c_2 \log n, \end{aligned}$$

i.e., there exists a constant $c_2 \in \mathcal{N}$, such that, for large enough n, for $t - l < n - (c_2 + 1) \log n$, $K(x) < n - \log n$ which is a contradiction.

Using this coding the condition $t - l < n - (c_2 + 1) \log n$, is sufficient. If $t - l \geq n - (c_2 + 1) \log n$ (i.e., $m = n - t + l < (c_2 + 1) \log n$), then $t > n - (c_2 + 1) \log n$, i.e., there is a right overlap of length $t - m > n - 2(c_2 + 1) \log n$. For large enough n, $n - 2(c_2 + 1) \log n > \frac{t}{2} = \frac{n - (c_2+1) \log n}{2}$, i.e., we fall in Case 5.

- Case 3): There is no right overlap and a left overlap of a in x^j and s, of length $0 \leq l < \frac{t}{2}$. (i.e., if $l = 0$ there is no overlap at all and $a_{t+1} = v_1$ in the figure.) (See Figure 3.)

  The proof is done in the same way as the first subcase of case 2.

- Case 4): There is a right overlap of a in x^j and s, of length $l \geq \frac{t}{2}$. (See Figure 4.)

  The right overlap covers bits $a_{t-l+1} \ldots a_t$. We first observe that since we have assumed that $v_1 \neq u_1$, the overlap must contain at least two

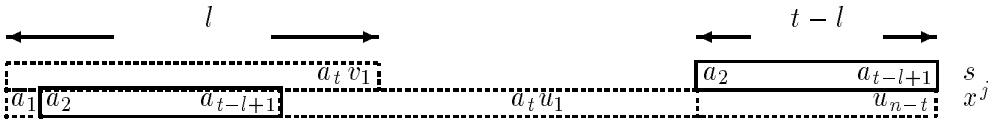

Figure 5: $l \geq \frac{t}{2}$

elements. This implies that the substring $a_2 \ldots a_{t-l}$ is repeated at least $\lfloor \frac{t-1}{t-l-1} \rfloor + 1$ times. (We can encode this with $O\left(\log\left(\left\lfloor \frac{t-1}{t-l-1} \right\rfloor + 1\right)\right)$ bits.) The encoding is given by a_1, plus $\lfloor \frac{t-1}{t-l-1} \rfloor + 1$ times $a_2 \ldots a_{t-l}$, plus a string x' of $n - 1 - \left(\left(\lfloor \frac{t-1}{t-l-1} \rfloor + 1\right)(t-l-1)\right)$ bits, plus the starting positions.

$$\begin{aligned} K(x) &\leq O(\log n) + t - l - 1 - (t-l-1) + n - 1 \\ &- \left(\left(\left\lfloor \frac{t-1}{t-l-1} \right\rfloor\right)(t-l-1)\right) \\ &\leq n + O(\log n) - \frac{l}{2}. \end{aligned}$$

Since $l \geq \frac{t}{2}$, there exists a constant $c_4 \in \mathcal{N}$, such that, for large enough n, for $t > 4(c_4+1)\log n$, $K(x) < n - \log n$ which is a contradiction.

- Case 5): There is a left overlap of a in x^j and s, of length $l \geq \frac{t}{2}$. There might be or not a right overlap. (See Figure 5 for the case of the right overlap.)

  The overlap covers the bits $a_1 \ldots a_l$. This means that the last $t-l$ bits are $a_2 \ldots a_{t-l+1}$. If $t < n - (t-l)$ and $(t-l+1) \geq \lfloor \frac{t}{2} \rfloor$ (i.e., the string $a_1 \ldots a_{t-l+1}$ is repeated once in $a_1 \ldots a_t$), or $t \geq n - (t-l)$ and $(t-l+1) \geq \lfloor \frac{n-(t-l)}{2} \rfloor$ (i.e., the string $a_1 \ldots a_{t-l+1}$ is repeated once in $a_1 \ldots a_{n-(t-l)}$), then the encoding is given by a_1, plus $a_2 \ldots a_{t-l+1}$ in positions 2 and $n-(t-l)+1$, that is

$$\begin{aligned} K(x) &\leq O(1) + n - (t-l) + O(\log n) \\ &\leq n - \lfloor \frac{t}{2} \rfloor - 1 + c_5 \log n, \end{aligned}$$

  i.e., there exists a constant $c_5 \in \mathcal{N}$, such that, for large enough n, for $t > 2(c_5+1)\log n$, $K(x) < n - \log n$ which is a contradiction.

If $t < n-(t-l)$ and $(t-l+1) < \lfloor \frac{t}{2} \rfloor$ (i.e., the string $a_1 \ldots a_{t-l+1}$ is repeated $\lfloor \frac{t}{t-l+1} \rfloor$ times in $a_1 \ldots a_t$), or $t \geq n-(t-l)$ and $(t-l+1) < \lfloor \frac{n-(t-l)}{2} \rfloor$ (i.e., the string $a_1 \ldots a_{t-l+1}$ is repeated $\lfloor \frac{n-(t-l)}{t-l+1} \rfloor$ times in $a_1 \ldots a_{n-(t-l)}$), then the encoding is given by $\lfloor \frac{t}{t-l+1} \rfloor$ (respectively $\lfloor \frac{n-(t-l)}{t-l+1} \rfloor$) times the string $a_1 a_2 \ldots a_{t-l+1}$ plus the remaining substring. With some calculations we obtain:

$$K(x) \quad \leq \quad n - \frac{t}{2} + c_6 \log n,$$

i.e., there exists a constant $c_6 \in \mathcal{N}$, such that, for large enough n, for $t > 2(c_6+1)\log n$, $K(x) < n - \log n$ which is a contradiction, or

$$K(x) \quad \leq \quad n - \frac{n-(t-l)}{2} + O(\log n) \leq \frac{n}{2} + c_7 \log n,$$

i.e., there exists a constant $c_7 \in \mathcal{N}$, such that, for large enough n, $K(x) < n - \log n$ which is a contradiction.

The constant c' of the statement of the lemma is then $c' = \max\{c_1, \ldots, c_7\}$. This ends the proof of the Lemma 11. ■

To prove the correctness of phase 3 we need another lemma which ensures that, if a string is Kolmogorov random, then there must exist a unique maximal substring:

LEMMA **12** : *Given a string $x = x_1 x_2 \ldots x_n$ of length n, and an integer $t \leq n$, there exist a constant $c'' \in \mathcal{N}$, such that, if n is large enough, for each $t \geq c'' \log n$, if $|S_x^t| < n$, then x is not Kolmogorov random.*

PROOF Let us take two substrings s and s' of x which are equal. The proof is very similar to the one of Lemma 11. There are two cases:

- Case 1): The (right or left) overlap l of s and s' in x is of length $> \frac{t}{2}$. Rotate the string x in such a way that the corresponding x^j starts with one of the substrings (w.l.o.g. s), and has a right overlap of length $> \frac{t}{2}$. (There might even be a left overlap, but we will not consider it.) (See Figure 6.)

  Since $t - l < \frac{t}{2}$, the string $s_1 \ldots s_{t-l}$ is repeated at least twice, more precisely $\lfloor \frac{t}{t-l} \rfloor$ times. The coding can then be: repeat $\lfloor \frac{t}{t-l} \rfloor$ times the

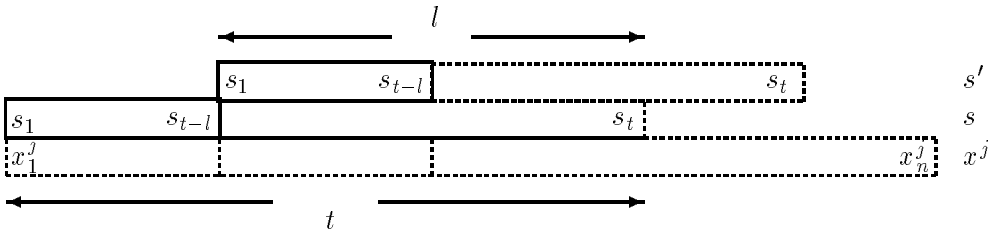

Figure 6: $l > \frac{t}{2}$

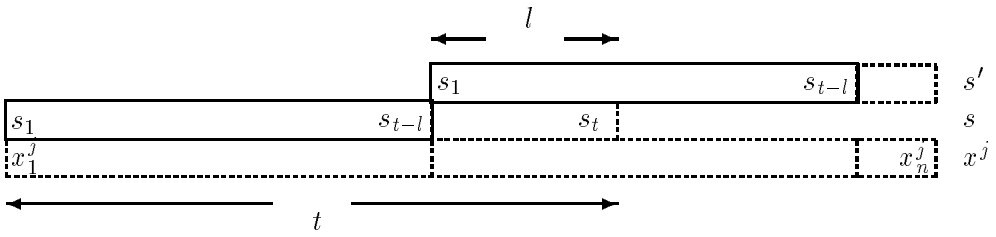

Figure 7: $l \leq \frac{t}{2}$

string $s_1 \ldots s_{t-l}$ and add the final string. This leads to:

$$\begin{aligned} K(x) &\leq n - \left(\left\lfloor \frac{t}{t-l} \right\rfloor - 1 \right) (t-l) + O(\log n) \\ &\leq n - \frac{t}{3} + c_8 \log n, \end{aligned}$$

i.e., there exists a constant $c_8 \in \mathcal{N}$, such that, for large enough n, for $t > 3(c_8+1)\log n$, $K(x) < n - \log n$ which is a contradiction.

- Case 2): Both the right and left overlap of s and s' in x are of length $\leq \frac{t}{2}$. Call l the biggest of the two and, w.l.o.g. assume that is the right overlap. Rotate the string x in such a way that the corresponding x^j starts with one of the substrings (w.l.o.g. s) and has l as a right overlap. (See Figure 7.)

  Since the right overlap is of length at most l, and the left overlap is $\leq l$, there are at least $t-l$ bits of s' that overlap the last bits of x^j. The encoding for $l > 0$, can be given as repeating twice the string $s_1 \ldots s_{t-l}$ from position 1 and $t-l+1$ for $l > 0$. This implies:

$$\begin{aligned} K(x) &\leq n - (t-l) + O(\log n) \\ &\leq n - \frac{t}{2} + c_9 \log n, \end{aligned}$$

i.e., there exists a constant $c_9 \in \mathcal{N}$, such that, for large enough n, for $t > 2(c_9+1)\log n$, $K(x) < n - \log n$ which is a contradiction.

If $l = 0$, i.e., there is no right or left overlap, the entire string s is repeated twice, that is:

$$K(x) \quad \leq \quad n - t + c_{10}\log n,$$

i.e., there exists a constant $c_{10} \in \mathcal{N}$, such that, for large enough n, for $t > (c_{10}+1)\log n$, $K(x) < n - \log n$ which is a contradiction.

The constant c'' of the statement of the lemma is then $c'' = \max\{c_8, c_9, c_{10}\}$. This ends the proof of Lemma 12. ■

Lemma 12 gives a necessary condition for a string to be Kolmogorov random. The next lemma will define a method to reconstruct a string starting from its set of substrings, and will prove that this string is unique.

LEMMA **13** *: A Kolmogorov random string, can be reconstructed from the set $S_x^{t'}$, where $t' > t$ and t is the smallest integer that satisfies the conditions of Lemma 12.*

PROOF We reconstruct the string from $S_x^{t'}$ by using the t length substrings. Let s be in $S_x^{t'}$, $s = s_1 \ldots s_t$, and let us add a new bit at the time by shifting of one position to the left and searching among the remaining t' length strings. If at each time there is a unique choice from the set $S_x^{t'}$, then the resulting string of length n must be unique. On the other hand let us assume that, w.l.o.g. at step 2 we can choose between two strings of length t', $s_2 \ldots s_t 0$ or $s_2 \ldots s_t 1$. If this happens this implies that, in the set $S_x^{t'-1}$ there are two strings that are equal, i.e., $|S_x^{t'-1}| < n$. But $t' - 1 \geq t$ where t satisfies the condition of Lemma 12, that is we are contradicting the incompressibility of x. ■

If the input string is x then for Lemma 12 and Lemma 13 there exists a unique leader since only one processor sees a maximal substring. The bit complexity of this step is then $O(\sum_{i=0}^{\log n} 2^i(i+1)) + O(n) = O(n\log n)$. If the input string is not x, then there are at most $\frac{n}{c\log n}$, $c \in \mathcal{N}$, that see the maximal string and that send a counter. In this case the bit complexity is still $O(n\log n) + O(n) = O(n\log n)$ since each counter will stop after reaching one of these processors, i.e., at most $c\log n$ bits message will travel for n steps. The correctness of this step is straightforward. If the input string is x, then there is a unique leader that acknowledges the fact that each processor has seen an acceptable neighborhood, and Lemma 11 states

that this condition is sufficient. On the other hand, if there is more then one leader, or at least one processor finds an unacceptable neighborhood (and from Lemma 11, if x is not the input string then this must happen), then a halt is sent and eventually all the processors will end their computation.

The overall bit complexity of these phases is $O(n \log n)$. These completes the proof of Theorem 9. ■

3 Lower Bounds

In this section we prove lower bounds for the string recognition problem. In the case of Kolmogorov random string we need only the lower bound of $\Omega(n \log n)$ of [10] and the equivalent boolean function formulation of the string recognition problem stated in the introduction. This lower bound implies the optimality of the algorithm given above. For the case of $x = w^t v$ we have the following;

THEOREM **14** *Let $x = w^{\lfloor \frac{n}{k} \rfloor} v$ be a string of period k such that $|x| = n$, $|w| = k$ and v is a proper prefix of w. The lower bound for recognizing any n-ary string on an n processors anonymous oriented ring is $\Omega(\frac{n^2}{k} + n \log n)$ if $v = \emptyset$, $\Omega(n \log n)$ if $v \neq \emptyset$.*

PROOF Let x be a string as in the statement of the theorem. The $\Omega(n \log n)$ bound follows from [10]. Hence we only need to prove the $\Omega(\frac{n^2}{k})$ lower bound for the case $v \neq \emptyset$.

By Theorem 5.1 in [1] we know that a lower bound on the bit complexity of computing a function can be derived as follows.

Given two input configurations y and z of length n, every algorithm that computes the function on input y requires at least

$$\sum_{i=0}^{\alpha} SI(y, i), \tag{1}$$

messages in the worst case.

Here α is a constant, such that there exist two processors p and q in the ring, for which p has the same α-neighborhood in y as q in z, but their output (at the end of the algorithm) is different. $SI(y, i)$ is the symmetry index function of the string y, i.e., the minimum number of elements in a equivalent i-neighborhood for processors in the configuration y.

Consider the string recognition problem with an input string $x = w^{\lfloor \frac{n}{k} \rfloor}$, $v = \emptyset$ and run the algorithm with input configurations $y = x$ and $z =$

$w^{\lfloor \frac{n}{k} \rfloor - 1} w_0 \ldots w_{k-1} \overline{w_k}$. There exist two processors p in y and q in z, that have the same $\lfloor \frac{n}{4} \rfloor - 1$-neighborhood, but their output for the problem is different since $y = x$, and $z \neq x$. On input y there are at least $\frac{\lfloor \frac{n}{k} \rfloor}{2}$ processors who see the same $\lfloor \frac{n}{4} \rfloor - 1$-neighborhood, we have then:

$$\sum_{i=0}^{\alpha} SI(y,i) = \sum_{i=0}^{\lfloor \frac{n}{4} \rfloor - 1} \frac{\lfloor \frac{n}{k} \rfloor}{2} = \lfloor \frac{n}{2} \rfloor \frac{\lfloor \frac{n}{k} \rfloor}{2}, \tag{2}$$

i.e., the lower bound is $\Omega(\frac{n^2}{k})$. ■

We finally show a family of strings whose lower bound is tight within a logarithmic factor. For every k we choose the string $x = (0^{k-1}1)^{\lfloor \frac{n}{k} \rfloor}$. The lower bound is $\Omega(nk + \frac{n^2}{k} + n \log n)$, since $\Omega(\frac{n^2}{k} + n \log n)$ is obtained from Theorem 14, and from the following intuition which adds an $O(nk)$ factor: given as input string x and $x' = (0^{k-1}1)^{\lfloor \frac{n}{k} \rfloor - 1} 0^k$, every processor must trivially wait nk messages on x' before one processor can detect that the string $x' \neq x$, and then broadcast it.

4 Conclusions and open problems

In this paper we have studied the string recognition problem on anonymous, asynchronous, n-processor rings.

We have shown a $\Theta(n \log n)$ bit complexity for Kolmogorov random strings, which in turns implies this same bit complexity for almost all strings. For strings in the form $x = w^t v$, $t \geq 0$, we have shown an $O(nk + n \log n)$ upper bound when $v \neq \emptyset$ and an $O(nk + n^2 \min\{1, \frac{\log n}{k}\})$ when $v = \emptyset$. When $v = \emptyset$ we have also shown an $\Omega(\frac{n^2}{k} + n \log n)$ lower bound which shows that the upper bound is tight, within a logarithmic factor for $k = O(\sqrt{n})$. When $v \neq \emptyset$ the upper bound $O(nk + n \log n)$ is tight for $k = O(\log n)$.

An interesting open problem would be tighten the previous bounds in all cases.

As a final remark, we observe that the proof given for the string recognition problem when x is a Kolmogorov random string can be modified and extended to other networks like tori, meshes, etc.

References

[1] H. Attiya, M. Snir and M. Warmuth, "Computing on an Anonymous Ring", Journal of the ACM, **35 (4)**, 845-875, (1988).

[2] P. W. Beame and H. L. Bodlaender, "Distributed Computing on Transitive Networks: The Torus", 6th Annual Symposium on Theoretical Aspects of Computer Science, STACS, B. Monier and R. Cori, editors, SVLNCS, 294-303, (1989).

[3] H. L. Bodlaender, S. Moran, M. K. Warmuth, "The Distributed Bit Complexity of the Ring: from the Anonymous to the Non-Anonymous Case", Information and Computation, **108 (1)**, 34,50, (1994).

[4] P. Ferragina, A. Monti and A. Roncato, "Trade-off between Computation Power and Common Knowledge in Anonymous Rings", Pre-Proceedings of Colloquium on Structural Information and Communication Complexity, May 16-18 (1994), Ottawa, Canada.

[5] O. Goldreich, L.Shrara, "On the Complexity of Global Computation in the Presence of Link Failures: the Case of Ring Configuration", Distributed Computing, **5, (3)**, 121-131, (1991).

[6] E. Kranakis and D. Krizanc, "Distributed Computing on Anonymous Hypercube Networks", Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, Dallas, Dec. 2-5, 722-729, (1991).

[7] E. Kranakis and D. Krizanc, "Computing Boolean Functions on Cayley Networks", Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing, Arlington, Texas, Dec. 1-4, 222-229, (1992).

[8] M. Li, P.M.B. Vitanyi, "Kolmogorov Complexity and its Applications", Handbook of Theoretical Computer Science, Algorithms and Complexity. The MIT Press, Cambridge, Massachusetts, (1990).

[9] H. Attiya and Y. Mansour, "Language Complexity on the Synchronous Anonymous Ring", Theoretical Computer Science, **53** 169-185, (1987).

[10] S. Moran and M. K. Warmuth, "Gap Theorems for Distributed Computation", in Proc. 5th ACM Symp. on Principles of Distributed Computing 131-140 (1986).

[11] A.K.Zvonkin, L.A. Levin, "The Complexity of Finite Objects and the Development of the Concepts of Information and Randomness by means of the Theory of Algorithms", Russian Math. Service, **25** 83-124, (1970).