On recognizing a string on an anonymous ring * † ‡

Evangelos Kranakis[§], Danny Krizanc[††] and Flaminia L. Luccio[**]

[§] School of Computer Science, Carleton University,

Ottawa, ON, K1S 5B6, Canada

kranakis@scs.carleton.ca

[††] Mathematics Department, Wesleyan University,

Middletown, CT, 06459, U.S.A.

krizanc@wesleyan.edu

[**] Dipartimento di Scienze Matematiche, Università di Trieste

via Valerio 12, 34127, Trieste, Italy

luccio@mathsun1.univ.trieste.it

*Research supported in part by National Sciences Engineering Research Council of Canada grant and by MURST, Progetto "Metodi e Problemi in Analisi Reale".

†A preliminary version of this paper appeared in the *Proceedings of the 20-th International Symposium, MFCS'95*, Springer Verlag LNCS, **969**, 392-401, (1995).

‡Please, send each communication related to this paper to: Dr.ssa Flaminia L. Luccio, Dipartimento di Scienze Matematiche, Università degli Studi di Trieste, via Valerio 12, 34127, Trieste, Italy, e-mail: luccio@mathsun1.univ.trieste.it.

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. In this paper we give new upper and lower bounds on the bit complexity of string recognition. For periodic strings, near optimal bounds are given which depend on the period of the string. For Kolmogorov random strings an optimal algorithm is given. As a consequence, 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.

1 Introduction

There have been several studies in the literature concerning the construction of communication efficient algorithms for computing functions on anonymous rings [1, 2, 4, 5, 6, 11], as well as on more general anonymous networks, like tori [3], hypercubes [7], Cayley networks [8], 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 [2] showed that all the functions that are computable on an anonymous asynchronous ring of processors, can be computed using $O(n^2)$ messages (one bit message if the functions are boolean) using input collection. They also proved that every algorithm that computes the minimum of all inputs (the OR function in the boolean case), requires $\Omega(n^2)$ messages, i.e., the bounds in this case are tight. Furthermore, they showed that almost all boolean functions on n variables require $\Omega(n^2)$ bits to compute. On the other hand, Moran and Warmuth [11] gave an $\Omega(n \log n)$ lower bound for computing any non-constant function and presented functions (derived from a recursive application of de Bruijn sequences) with an $O(n \log n)$ 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.

1.1 Results of the paper

Let $x = x_0x_1 \cdots x_{n-1}, y = y_0y_1 \cdots y_{n-1} \in \{0,1\}^n$ be binary strings of length n and suppose that for each $i = 0, \ldots, n-1$ the i-th processor of an anonymous ring knows the i-th bit y_i of y as well as the entire string x. In the **string recognition problem** all the n processors recognize whether the input string y is a cyclic shift of the string x or not.

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 cyclic shifts, the function f_x is computable in the oriented anonymous ring [2]. We are interested in determining the bit complexity of computing $f_x(y)$.

The input collection algorithm of Attiya, Snir and Warmuth [2] 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 independent of x and y. In this paper, we provide more efficient algorithms by exploiting periodicities. Let x have period k, i.e., $x = w^t v$, $t \geq 1$, $|w| = k$, v is a prefix of w, and k is minimum. If $v = \emptyset$, the null string, then we give an $O(nk + \frac{n^2}{k})$ upper bound and $\Omega(n \log n + \frac{n^2}{k})$ lower bound. If $v \neq \emptyset$ then we give an $O(nk + n \log n)$ upper bound and $\Omega(n \log n)$ lower bound.

Further, we show 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 been used to show lower bounds. From an intuitive point of view, Kolmogorov complexity theory deals with the amount of information that a string contains [10, 13].

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.

1.2 Preliminaries

As our model of computation we consider the standard anonymous, asynchronous, ring of n processors [2]. The size n of the ring is known to the processors, and initially each processor is given a single bit of y as well as a string x of length n. By anonymity, we mean that the processors execute the same algorithm given the same data. At the end of the computation, it is desired that all the processors determine correctly whether or not the input string y is a rotation of the input string x.

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 neighbour. 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 i-th processor. If y_p is the input to processor p then $\langle y_p : p < n \rangle$ denotes the network input. The right (respectively, left) k-neighbourhood of processor p is the sequence of bits $\langle y_{p+i} : 0 \leq i < k \rangle$ (respectively, $\langle y_{p-i} : 0 \leq i < k \rangle$), where the operations $+, -$ (as well as all the other later defined operations on strings) are performed modulo n.

Before we proceed with the description of our algorithms we give some necessary definitions.

Definition 1.1 *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. Given a string $x = x_0x_1 \ldots x_{n-1}$ of length n, let $x \uparrow s = x_0x_1 \ldots x_{s-1}$ denote the first s bits of x, and $x(r) = x_rx_{r+1} \ldots x_{n-1}x_0x_1 \ldots x_{r-1}$ the r-th rotation of x. S_x^t is the set of all the different length t, $t \leq n$, substrings of $x(r)$, $r = 0, \ldots, n-1$, i.e., $S_x^t = \{x(r) \uparrow t \,|\, r = 0, \ldots, n-1\}$.*

Definition 1.2 *Given a string x of length n, a programming language S, let $K_S(x)$ denote the length of the minimum program that outputs x using S, i.e., $K_S(x) = \min\{|p| : p$ is a program in S, and $S(p) = x\}$.*

From now on $K(x)$ refers 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 *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

We now give an efficient algorithm for the string recognition problem.

Theorem 2.1 *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 $O(nk + \frac{n^2}{k})$ if $v = \emptyset$, and $O(nk + n \log n)$ if $v \neq \emptyset$.*

PROOF The proof of the theorem consists of an efficient input collection algorithm satisfying the requirements above. Recall that string x is known to all the processors, and let y_p be the input to processor p. The algorithm determines whether or not x is a rotation of the input $y_0 y_1 \ldots y_{n-1}$. Each processor p executes the following algorithm which is described in several phases.

Phase 1 Processor p 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 $t \geq 1$. This is a local step executed initially by all the processors and does not contribute 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. Processor p now computes $\langle y_{p+i} : 0 \leq i < 2k+1 \rangle$ and $\langle y_{p-i} : 0 \leq i < k \rangle$ by executing the following algorithm.

/* computing the right $(2k+1)$-neighbourhood */ send y_p to the left;

for $i = 1$ to $2k$ **do** send bit received from the right to the left **od**;

/* computing the left k-neighbourhood */ send y_p to the right;

for $i = 1$ to $k-1$ **do** send bit received from the left to the right **od**;

It will become apparent in the case $v \neq \emptyset$ (discussed in the sequel) why we need to compute a $(2k+1)$-neighbourhood to the right and a k-neighbourhood to the left. For the time being we will only use the left k-neighbourhood. Clearly, the bit complexity of Phase 2 is $O(nk)$.

Phase 3 Processor p executes the main part of the algorithm that determines 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 $v = \emptyset$: During this phase processors send and receive three types of messages: "yes", "no" and "stop", where "yes" indicates tentative agreement between the input y and x, "no" indicates certain disagreement between between the input y and x, and "stop" indicates certain agreement between the input y and x.

Let $U_p = \langle y_{p-i} : 0 \leq i < k \rangle$ be computed by processor p in Phase 2. Processor p checks if U_p is a rotation of w and if not it sends a "no" and stops. If on the other hand it finds out that U_p is the lexicographically maximal rotation of w it becomes a *leader* and sends a "yes" message around the ring.

If a leader receives $\lfloor \frac{n}{k} \rfloor$ "yes"s then it broadcasts a "stop" indicating that y is a rotation of x and stops, otherwise it broadcasts a "no" message and stops.

During the confirmation stage at most $\lfloor \frac{n}{k} \rfloor$ processors participate (the leaders, i.e., the ones that have a lexicographically maximal rotation of w); each of these processors sends a "yes" around the ring (confirming that it has a lexicographically maximal rotation of w) and waits for $\lfloor \frac{n}{k} \rfloor$ "yes"s or a "no." At most $\lfloor \frac{n}{k} \rfloor$ "yes"s are sent and each travels at most n steps. At most 1 stop message is sent and travels n steps. Finally, if any processor either has a k-neighbourhood which is not a rotation of w or receives a "no" message, it

sends it to its neighbour and does not transmit again. Thus the overall bit complexity of this phase is $O(\frac{n^2}{k})$. (The "yes", "no", and "stop" messages require only $O(1)$ bits.)

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 y 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 2.1.

If there exists a processor $p \in \{i = 0, \ldots, n-1\}$ that does not see a rotation of the input, this obviously implies $y \neq w^t$, and p will broadcast a message to stop the other processors.

If all the processors have seen a rotation of the input, then $y = w^t$ must hold. Consider $w = w(0) = w_0 w_1 \ldots w_{k-1}$ and let us assume to the contrary that there exists a processor p whose left k-neighbourhood $\langle y_{p-i} : 0 \leq i < k \rangle$ is the string $w(l) = w_l \ldots w_{k-1} w_0 \ldots w_{l-1}$, where $l \in \{0, \ldots, k-1\}$, but the k-neighbourhood of processor $p+1$ is the string $w(s) = w_s \ldots w_{k-1} w_0 \ldots w_{s-1}$, where $s \in \{0, \ldots, n-1\}$, and $w(s) \neq w(l+1)$. Since p and $p+1$ are adjacent processors, their left k-neighbourhoods must have $k-1$ bits that are the same, i.e., $w_{l+1} \ldots w_{k-1} w_0 \ldots w_{l-1} = w_s \ldots w_{k-1} w_0 \ldots w_{s-2}$. Moreover, if $w(s) \neq w(l+1)$, then $w_l \neq w_{s-1}$, and since we are working with binary strings, we have $w_l = \overline{w_{s-1}}$.

Observe now that every rotation $w(r) = w_r \ldots w_{k-1} w_0 \ldots w_{r-1}$, $r = 0, \ldots, k-1$, of the binary string $w(0) = w_0 \ldots w_{k-1}$ of length k has the same number of ones, therefore the assumption that two rotations of w have a different number of ones (since $w_l \neq w_{s-1}$), leads to a contradiction. If each processor sees a rotation of the input, then the string can only be $y = w^t$.

CASE $v \neq \emptyset$: Assume that $x = x_0 \dots x_{n-1} = w^t v$, and $v = v_0 \dots v_{l-1}$ is a proper prefix of w of length l, i.e., $v = w_0 \dots w_{l-1}$. We can easily prove the following:

Lemma 2.1 *There exists a substring $z = z_0 \dots z_{k-1}$ of length k of x that has weight (i.e., number of bits equal to 1) different from the weight of w.*

PROOF The proof is straight forward and uses the fact that k is the period of x and $v \neq \emptyset$.

□

Let $V_p = \langle y_{p-i} : 0 \leq i < k \rangle \bigcup \langle y_{p+i} : 0 \leq i < 2k+1 \rangle$ be computed by processor p in Phase 2. In view of the representation of x in the form $w^t v$ it is an immediate consequence of Lemma 2.1 that there is a *unique* processor in the string x with a right k-neighbourhood that is not a rotation of w, but such that the k-neighbourhoods of all k processors to its left are indeed rotations of w and the $k-1$ processors to its right have the *correct* k-neighbourhoods, i.e., substrings of the string wvw.

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 $k-1$ processors to its right have the correct k-neighbourhoods, and in addition all other processors have a k-neighbourhood 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 2.2 *If after Phase 2 of the algorithm there exists at least one processor whose $3k$-bits string obtained in Phase 2 is different from $3k$ bits of the string $w^3 v w^3$, then $y \neq w^{\lfloor \frac{n}{k} \rfloor} v$. If on the other hand all the processors have a $3k$-neighbourhood that is a substring of $w^3 v w^3$, then $y = w^{\lfloor \frac{n}{k} \rfloor} v$, or $y = w^{s_1} v w^{s_2} v \dots w^{s_i} v$, for $s_j \geq 2$, $j = 1, \dots i$, $i \geq 2$.*

PROOF The first part of the proof is straightforward. For the second part let us now assume that during Phase 2 two neighbouring processors obtain two $3k$-bit strings that are substrings of w^3vw^3, and let us denote them by $Z_1 = z_1 \dots z_{3k}$ and $Z_2 = z_2 \dots z_{3k+1}$. We will now show that the only strings that can occur are $y = w^{\lfloor \frac{n}{k} \rfloor} v$, or $y = w^{s_1} v w^{s_2} v \dots w^{s_i} v$, for $s_j \geq 2$, $j = 1, \dots, i$, $i \geq 2$. When we try to build up a string under the restriction that all processors have a correct $3k$-neighbourhood of x, four different cases arise depending on Z_1 and Z_2:

1. Z_1 and Z_2 are both rotations of w^3.

2. Z_1 is a rotation of w^3 and Z_2 is a rotation of $3k$ bits of w^3vw^3, but $Z_2 \neq w^3$.

3. Z_1 is a rotation of $3k$ bits of w^3vw^3, but $Z_1 \neq w^3$, and Z_2 is a rotation of w^3.

4. Both Z_1 and Z_2 are rotations of $3k$ bits of w^3vw^3, but $Z_1, Z_2 \neq w^3$.

For each of the cases it is easy to show that either the resulting string is x or it contains at least two substrings v, with all v's separated by at least a w^2, therefore proving the lemma. (For more details of the proof refer to [9].) □

It is now easy to convert this characterization into an $O(nk + n \log n)$ algorithm for determining whether or not the input string y is a rotation of x. The first two phases are the ones previously described. During Phase 3 processors may receive one of three types of messages: "no" and "stop" messages as above or a "counter" message containing a value c which they increase by one and then forward.

Initially every processor p checks if $V_p = \langle y_{p-i} : 0 \leq i < k \rangle \bigcup \langle y_{p+i} : 0 \leq i < 2k+1 \rangle$ is a substring of w^3vw^3 and if not broadcasts a "no" and stops.

If on the other hand p finds out from V_p that it is in the unique position, it becomes a *leader* and sends a counter c with initial value 0 around the ring.

If a leader receives a counter with $c < n$ then it broadcasts a "no" message, otherwise (if $c = n$) a "stop" message, and in both cases it stops.

The observations made above prove the correctness of the algorithm. Only one processor can be in the unique position. If there is more than one, this situation is detected by another processor in a unique position, by receiving $c < n$. Note that it is not possible that every processor has w^3 as a neighbourhood since $|x| = n$ cannot be divided by $|w|$, (since $|v| < |w|$). The bit complexity of the algorithm is straightforward. In the worst case in last phase, 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 therefore $O(nk + n \log n)$. This completes the proof of Theorem 2.1. □

We now present an algorithm that optimally recognizes a Kolmogorov random string (i.e. almost every string) using $\Theta(n \log n)$ bits. (The lower bound follows from [11].)

Theorem 2.2 *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 complexity $\Theta(n \log n)$.*

PROOF As before, the input to any processor p consists of the string x and the bit y_p. In the algorithm we use a constant c that will be fixed more precisely below. Lemmas 2.3 and 2.4 below imply that, if the $c\lceil \log n \rceil$-neighbourhood of a processor coincides with one of the strings contained in $S_x^t = \{x(r) \uparrow t \mid r = 0, \ldots, n-1\}$ (see Section 1.2), then, for n large enough, the input string y must be x or one of its rotations.

Let $c = \max\{c', c''\}$, where c' and c'' are constants given in Lemma 2.3 and 2.4 below. Processor p executes the following phases.

Phase 1 Compute and store S_x^t, for $t = c \log n$. This is a local step and it does not contribute to the overall bit complexity of the final algorithm.

Phase 2 Compute $W_p = \langle y_{p-i} : 0 \leq i < t \rangle$ as in Phase 2 of algorithm above; i.e.,

/*computing the left t-neighbourhood*/ send y_p to the right;

for $i = 1$ to $t - 1$ **do** send bit received from the left to the right **od**;

then check whether the t-neighbourhood coincides with one of the strings contained in S_x^t. The cost of this phase, in terms of bits transmitted, is $O(n \log n)$.

Phase 3 Processor p determines if the input string y 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, then the input string y must be x (see Lemma 2.4), otherwise, if at least one finds a different substring, it is not x. Now processor p executes the following:

if $W_p \notin S_x^t$ **then** broadcast a "no" message and stop

else if W_p is the maximal string in S_x^t

then send a counter c around the ring; /* counter c is only incremented by processors who have seen an acceptable string */

if a counter value $c = n$ is received

then broadcast a "stop" message and stop

else /* $c < n$, or another processor has seen a not acceptable string and has sent a stop */ broadcast a "no" message and stop

fi

else /* W_p is not a maximal string */

if an unacceptable string is seen

then broadcast a "no" message and stop

else **if** a counter c is received **then** broadcast $c := c + 1$

else /*a "no" or a "stop" is received*/ broadcast the message

and stop **fi**

fi

fi

fi

We can now precisely define a method to find the right value of c that justifies the correctness of Phases 2 and 3 of the algorithm.

The following lemmas are easily derived from the results in [12]. For a complete proof see [9].

Lemma 2.3 *Given a string $x = x_0x_1 \ldots x_{n-1}$ of length n, and an integer $t \leq n$, there exists 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.*

Lemma 2.4 *There exists a constant $c'' \in \mathcal{N}$, such that, for n sufficiently large and for all $t \geq c'' \log n$, for any two strings $x = x_0x_1 \ldots x_{n-1}$ and $y = y_0y_1 \ldots y_{n-1}$ of length n, if x is Kolmogorov random, and $S_y^t \subseteq S_x^t$, then $x = y$ up to rotation.*

If the input string y is x then from Lemma 2.3 there exists a unique leader since only one processor sees a maximal substring. Therefore the bit complexity in this case

is $O(\sum_{i=0}^{\log n} 2^i(i+1)) + O(n) = O(n \log n)$. If the input string y is not x, then there are at most $\frac{n}{c \log n}$ processors, 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 the next of these processors, i.e., $c \log n$ bit messages will travel for at most n steps. The correctness of the algorithm is straightforward. If the input string y is x, then there is a unique leader that acknowledges the fact that each processor has an acceptable neighbourhood, and Lemma 2.4 states that this condition is sufficient. On the other hand, if there is more than one leader, or at least one processor has an unacceptable neighbourhood (and from Lemma 2.4, if the input string y is not x then this must happen), then a "no" is sent and eventually all the processors will end their computation.

The overall bit complexity of these phases is $O(n \log n)$. This completes the proof of Theorem 2.2. □

3 Lower Bounds

In this section we prove lower bounds for the string recognition problem. In the case of Kolmogorov random strings we need only the lower bound of $\Omega(n \log n)$ of [11] 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 3.1 *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. A lower bound for recognizing any n-ary string on an n processors anonymous oriented ring is $\Omega(n \log n + \frac{n^2}{k})$ if $v = \emptyset$, $\Omega(n \log n)$ if $v \neq \emptyset$.*

PROOF For both cases of v, the $\Omega(n \log n)$ lower bound follows from [11]. Hence we only need to prove the $\Omega(\frac{n^2}{k})$ lower bound for the case $v = \emptyset$.

By Theorem 5.1 in [2] 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 $\sum_{i=0}^{\alpha} SI(y,i)$ 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-neighbourhood 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 processors with an equivalent i-neighbourhood on input y.

Consider now the string recognition problem with an input string $x = w^{\lfloor \frac{n}{k} \rfloor}$, where $w = w_0, w_1, \ldots w_k$ and $v = \emptyset$, and run the algorithm with input configurations $z = w^{\lfloor \frac{n}{k} \rfloor - 1} w_0 \ldots w_{k-1} \overline{w_k}$, and $y = x$. There exist two processors p in y and q in z, that have the same $(\lfloor \frac{n}{4} \rfloor - 1)$-neighbourhood, 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 with the same $(\lfloor \frac{n}{4} \rfloor - 1)$-neighbourhood, so we have:

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

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

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 bound for Kolmogorov random strings, which in turns implies this same bit complexity for almost all strings. For strings, with period k, of the form $x = w^t v$, $t \geq 1$, $|w| = k$, we have shown an $O(nk + n \log n)$ bit upper bound when $v \neq \emptyset$ and an $O(nk + \frac{n^2}{k})$ bit upper bound when $v = \emptyset$. When $v = \emptyset$ we have also shown an $\Omega(n \log n + \frac{n^2}{k})$ bit lower bound which shows that the upper bound is tight, 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 is to tighten these 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.

Acknowledgments

The authors would like to thank Paul Vitányi for pointing out the usefulness of reference [12] in proving Lemma 2.3 and 2.4. Thanks also to Nicola Santoro for the idea on how to tighten the bounds in the case of periodic strings with $v = \emptyset$.

References

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

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

[3] P.W. Beame and H.L. Bodlaender, "Distributed Computing on Transitive Networks: The Torus", *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science, STACS*, Springer Verlag LNCS, **349**, 294-303, 1989.

[4] H.L. Bodlaender, S. Moran and 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).

[5] P. Ferragina, A. Monti and A. Roncato, "Trade-off between Computation Power and Common Knowledge in Anonymous Rings", *Proceedings of the Colloquium on Structural Information and Communication Complexity*, Ottawa, Canada, 35-48, 1994.

[6] O. Goldreich and 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).

[7] E. Kranakis and D. Krizanc, "Distributed Computing on Anonymous Hypercube Networks", *J. Algorithms*, **23, (1)**, 32-50, (1997).

[8] E. Kranakis and D. Krizanc, "Computing Boolean Functions on Cayley Networks", *Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing*, Arlington, Texas, U.S.A., 222-229, 1992.

[9] E. Kranakis, D. Krizanc and F. Luccio, "String Recognition on Anonymous Rings", Technical Report 256, School of Computer Science, Carleton University, Ottawa, Canada, 1994.

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

[11] S. Moran and M. K. Warmuth, "Gap Theorems for Distributed Computation", *SIAM Journal of Computing*, **22 (2)**, 379-394, (1993).

[12] J. Seiferas, "A Simplified Lower Bound for Context-free Language Recognition," *Information and Control*, **69**, 255-260, (1986).

[13] A.K. Zvonkin and 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 Mathematical Surveys*, **25**, 83-124, (1970).