ON THE BIT COMPLEXITY OF DISTRIBUTED COMPUTATIONS IN A RING WITH A LEADER

by

Y. Mansour and S. Zaks

Technical Report #422

June 1986
ON THE BIT COMPLEXITY OF DISTRIBUTED COMPUTATIONS IN A RING WITH A LEADER

Y. Mansour and S. Zaks

Department of Computer Science
Technion - Israel Institute of Technology
Haifa, Israel 32000

ABSTRACT

We study the bit complexity of pattern recognition in a distributed ring with a leader. Each processor gets as input a letter from some alphabet, and these concatenated letters, starting at the leader, form the pattern of the ring. The leader initiates an algorithm that accepts or rejects this pattern. Thus each algorithm recognizes a language over a given alphabet. We prove the following ($n$ is the size of the ring, not known a priori to any of the processors):

1. A language is recognized by an algorithm that uses $O(n)$ bits if and only if it is regular.
2. Every non-regular language requires at least $\Omega(n \log n)$ bits for its recognition (clearly, every language requires no more than $O(n^2)$ bits for its recognition).
3. For every function $g(n)$, $\Omega(n \log n) \leq g(n) \leq O(n^2)$, there is a language that requires $\Theta(g(n))$ bits for its recognition.

1 This is a revised version of report #383, November 1985, and appears in the Proceedings of the 5th ACM Symposium on Principles of Distributed Computing, Calgary, August 1986.
1. INTRODUCTION

Much research is recently concentrated on distributed computations, in attempts to understand their complexity. In these studies, a network of processors is given, and by exchanging messages they have to solve certain problems. The complexity measures studied are the message complexity and the bit complexity. A widely studied problem is identifying a unique processor, usually referred to as finding a leader. Also, even the existence of a leader still leaves many open questions regarding the communication complexity of certain problems, and this study belongs to this category.

We study the bit complexity of distributed pattern recognition in a ring with a leader. Each processor gets as input a letter, and the leader initiates an algorithm that accepts or rejects the pattern formed by the concatenation of these letters (starting at the leader). Thus, we can view an algorithm as computing a function or recognizing a language over a given alphabet. We study the communication complexity of this recognition problem.

Assuming a ring of size $n$, it is clear that every function can be computed with $n$ messages (the leader collects the information from all the processors and then computes the function locally). We assume that every processor participates in the algorithm; otherwise, since $n$ is not known, only trivial languages (of the form $L_1 \cdot L_2$, where $L_1$ and $L_2$ are finite languages), can be recognized with $O(1)$ messages; therefore $n$ is clearly a lower bound for the number of messages. For that reason, we consider bit complexity rather than message complexity. The leader can obtain all the information about all the processors in $O(n^2)$ bits, giving a trivial upper bound for the computation of every function. In this study we shed some light on this $\Theta(n) - \Theta(n^2)$ range.

We show that a language is recognized by an algorithm that uses $O(n)$ bits if and only if it is regular, and that every non-regular language requires at least $\Omega(n \log n)$ bits for its recognition. In other words, for every function $g(n)$, $\Omega(n) < g(n) < O(n \log n)$, there is no language that requires $\Theta(g(n))$ bits for its recognition. The hierarchy of the non-regular languages, in terms of their bit complexity, is not the natural one; we show two examples: a linear language that requires $\Omega(n^2)$ bits, and a context-sensitive language that is not context-free and can be recognized in $O(n \log n)$ bits. We show that for every function $g(n)$, $\Omega(n \log n) < g(n) < O(n^2)$, there is a language that requires $\Theta(g(n))$ bits for its recognition.

The problem of distributively finding a leader in a unidirectional ring with distinct identities appears in [DKR,P], and algorithms that use at most $O(n \log n)$ messages are shown; this bound is best possible ([PKR]). In [IR,ASW] it is proved that in the case when the identities are not distinct and the number of processors is unknown, there is no algorithm that can compute the ring size. Therefore, in order to be able to compute any function, we must either know the size of the ring or have a leader. The case where only the size of the ring is known is the main subject of [ASW] where - even though a leader cannot be found - functions concerning the pattern of the ring can still be computed, and results about the cost of this computation are discussed. Even the existence of a leader still leaves many open questions regarding the communication complexity of certain problems, and various studies in distributed graph algorithms make this assumption (see, for example, [E], [KRS] or [MC]). Another approach is discussed in [IR], where probability is used in order to break the symmetry (find a leader). In [MW], computation of functions on an asynchronous ring is studied, for the case were no leader exists, but the size of the ring is known to each.
processor. A gap is shown between the number of bits required to compute a constant function (zero bits) and any interesting (non-constant) function (Ω(n log n) bits).

A similar result for Turing machines computations is presented in [HA], [HE] and [T]. It is shown that a one-tape Turing machine recognizes a language in time \( O(n) \) if and only if the language is regular, and that a non-regular language requires at least \( \Omega(n \log n) \) time. We further discuss this result in the Summary Section.

The paper is organized as follows. In Section 2 we introduce the model and the notations. In Section 3 we prove the results for regular languages in a unidirectional ring. Section 4 deals with the lower bound for non-regular languages. In Section 5 we prove the results for regular languages in a bidirectional ring. Section 6 contains few notes concerning the bit-complexity hierarchy, and Section 7 contains a summary and open problems.

2. THE MODEL

The model we discuss is a distributed asynchronous ring of processors, with one specific processor (the leader). The number of processors is not known to any of the processors. Each processor holds one letter from a given alphabet. The processors can communicate only through the edges of the ring, and each message is assumed to have a finite transmission time. All the processors, excluding the leader, execute the same algorithm.

We deal with algorithms that recognize the pattern on the ring. We assume that the leader initiates the algorithm. [In the unidirectional case, the execution of each algorithm is unique, and can be described as a sequence of messages sent by the processors around the ring in a round-robin fashion, starting with the leader.] We choose to terminate the algorithm when the leader accepts or rejects the pattern on the ring; in other words, the leader wants to know whether the word on the ring belongs to a given language (clearly, the leader can then inform all the other processors that the algorithm terminated). We do not deal with languages that can be recognized with a fixed number of messages.

We use the following notations and definitions:

- \( n \): The size of the ring.
- \( \Sigma \): A finite set (the alphabet).
- \( \sigma \): An element of \( \Sigma \) (a letter).
- \( P_i \): The \( i \)-th processor, where \( P_1 \) is the leader and \( P_i \), \( 1 < i < n \), is the processor that can communicate with \( P_{i-1} \) and \( P_{i+1} \), \( P_1 \) can communicate with \( P_2 \) and \( P_n \), and \( P_n \) can communicate with \( P_1 \) and \( P_{n-1} \).

[In the unidirectional case, \( P_i \), \( i < n \), sends messages to \( P_{i+1} \) and \( P_n \) sends messages to \( P_1 \).

- \( \sigma_i \): The letter held in \( P_i \).
- \( w \): A word in \( \Sigma^* \).

We say that the ring is \textit{labeled} with \( w \) if \( w = \sigma_1 \sigma_2 \cdots \sigma_n \).
L is a language over Σ.

A is an algorithm that accepts or rejects the pattern on the ring.

The language recognized by A is the set of all words accepted by A. Two algorithms are equivalent if they recognize the same language.

exA is an execution of the algorithm A: We assume that no two messages are sent (or received) at the same time; therefore, an execution is the sequence of messages that the algorithm A sent and received. [In the unidirectional case the execution is unique.]

passA is a sequence of n messages, the first of which sent by the leader, that are sent during the execution of the algorithm A [this definition applies only for the unidirectional case].

πA is the maximal number of messages sent by any processor during any execution of the algorithm A. πA will be used only if such a number exists. [In the unidirectional case πA is also the maximal number of passes.]

MA is the set of messages used by the algorithm A (can be infinite).

BITA(n) is the bit complexity of the algorithm A. This means that for a ring labeled with w, |w| = n, A uses at most BITA(n) bits, on every possible execution.

When the algorithm A is clear from the text we use pass, ex and MA for passA, exA and MA, respectively. For terminology concerning formal languages and graph theory we follow [HU] and [E], respectively.

3. REGULAR LANGUAGES - UNIDIRECTIONAL RINGS

In this section we study the unidirectional rings, and show that there exists an algorithm A that recognizes L such that BITA(n) if and only if L is a regular language. We first show that the condition is necessary.

Theorem 1: Let L be a regular language. There exists a unidirectional algorithm A that recognizes L and satisfies BITA(n) = O(n).

Proof: Let FA = (Q, Σ, δ, q0, F) be a finite automaton that recognizes the language L. We construct an algorithm A that recognizes L in one pass. Initially, each processor in the ring will have a copy of FA, and the message that processor pi will send will contain the state of FA after scanning i letters, i = 1, 2, ..., n. This is done as follows: suppose the ring is labeled with w = σ1 · · · σn; then p1 (the leader) sends to p2 the message containing q1 = δ(q0, σ1), and in general, pi sends a message containing qi = δ(qi-1, σi), i = 1, ..., n. qn = δ(qn, w), so when p1 receives this message it can decide whether w ∈ L.

Each of the n messages require no more than ⌈ log |Q| ⌉ bits, hence BITA(n) ≤ ⌈ log |Q| ⌉ n = O(n), and this completes the proof. □
Theorem 2: Let $L$ be a language that is recognized by a one-pass unidirectional algorithm $A$, with $BIT_A(n) = O(n)$. Then $L$ is regular.

Proof: Let $A$ be a one pass unidirectional algorithm that recognizes the language $L$, such that $BIT_A(n) = O(n)$. Suppose $A$ is using the (possibly infinite) set of messages $M_A = \{m_1, \ldots, m_i, \ldots\}$. We build an infinite directed edge-labeled graph $G' = (V', E')$ that represents the way the algorithm works, as follows: $V' = \{v_0, v_1, \ldots, v_i, \ldots\}$, where $v_i$ represents a special message that makes the leader initiate the algorithm, and $v_i$ represents the message $m_i \in M$ for every $i$; $(v_i, v_j) \in E'$ is a directed edge from $v_i$ to $v_j$ with a label $l((e)) = \sigma$ if the following holds: when a processor has the initial value $\sigma$ and it receives the message $m_i$, it sends the message $m_j$ (here we use the assumption that all the processors execute the same algorithm).

From $G' = (V', E')$, we construct the subgraph $G = (V, E)$ induced by the vertices reachable from $v_0$. If the graph $G$ is finite, then by König's Infinity Lemma (see [2]) we obtain a simple infinite path $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots$. Every prefix of size $n+1$ of this path represents a ring of size $n$ labeled with the word $w = \sigma_1 \cdots \sigma_n$, where $\sigma_i = l((e))$. The algorithm $A$ will send on this ring the $n$ distinct messages $m_1, m_2, \ldots, m_n$. At least $O(n)$ of them will require $O(\log n)$ bits each, and therefore $BIT_A(n) = \Omega(n \log n)$. This contradicts the assumption that $BIT_A(n) = O(n)$. It follows that the graph $G$ is finite, and it clearly represents a state diagram of a finite automaton that recognizes $L$, hence $L$ is regular.

 Actually, by the last part of the proof, the following stronger results hold:

Corollary 1: Let $A$ be a unidirectional algorithm that recognizes a non-regular language in one pass. Then:

(a) $A$ uses an infinite number of messages, and

(b) $BIT_A(n) = \Omega(n \log n)$.

Corollary 2:

(a) Let $A$ be a one pass unidirectional algorithm that uses an infinite number of distinct messages. Then $BIT_A(n) = \Omega(n \log n)$.

(b) Let $A$ be any unidirectional algorithm. If there is a pass where all the processors, excluding the leader, have no knowledge other than their initial value $\sigma$, and if $A$ uses an infinite number of distinct messages in this pass, then the bit complexity of this pass is $\Omega(n \log n)$, hence $BIT_A(n) = \Omega(n \log n)$.

We now show the general unidirectional case.

Theorem 3: Let $L$ be a language that is recognized by a unidirectional algorithm $A$, with $BIT_A(n) = O(n)$. Then $L$ is regular.

Proof: Let $A$ be a unidirectional algorithm that recognizes the language $L$ such that, for some $c > 0$, $BIT_A(n) < cn$ for every $n$, so the maximal number of messages $\pi_A$ sent by any processor in any execution
satisfies \( \pi_A < c \). We construct an equivalent one pass algorithm \( A'' \). \( p_1 \) will send a message containing all possible sequences of \( \pi_A \) messages; we later show that \( M_A \) is finite. The processor \( p_i \) when receiving the message simulates its behavior on every sequence of \( \pi_A \) messages it receives, and sends a message containing all the resulting sequences of \( \pi_A \) messages. When the leader receives a message it can check which of the sequences of \( \pi_A \) messages is the one that it uses during the algorithm, and therefore \( A'' \) is equivalent to \( A \).

We now show that \( M_A \) is finite. We construct an equivalent algorithm \( A' \) that will not need any information about previous messages kept in the processors. \( A' \) will perform the first pass like \( A \). In the \( i \)-th pass \( p_1 \) will send all its previous \( i-1 \) messages plus the new message of pass \( i \). Each processor will be able to simulate its action in previous passes and send to the next processor all the \( i \)-messages consisting of the \( i-1 \) messages it sent during the previous \( i-1 \) passes and the new message of the \( i \)-th pass. Clearly, \( BIT_A(n) \leq \pi_A(c n) < c^2 n = O(n) \). \( A \) and \( A'' \) are equivalent, and both use an infinite number of messages or a finite number of messages. If \( M_A \) is infinite, then there is a pass in \( A' \) that uses an infinite number of messages, and by Corollary 2(b) \( BIT_A(n) = O(n \log n) \), which contradicts the fact that \( BIT_A(n) = O(n) \). \( M_A \) is therefore finite, which implies that \( M_A \) is finite. We conclude that \( BIT_A(n) \leq n \log M_A = O(n) \).

We have constructed a one pass algorithm \( A'' \) with a bit complexity of \( O(n) \). By Theorem 2 the language recognized by \( A'' \) is regular. But \( A \) and \( A'' \) are equivalent, hence \( L \) is regular. \( \square \)

4. NON-REGULAR LANGUAGES: UNIDIRECTIONAL AND BIDIRECTIONAL RINGS

In this section we prove the lower bound for bidirectional rings; the lower bound for unidirectional rings follows immediately.

Theorem 4: Let \( L \) be a non-regular language, and \( A \) an algorithm that recognizes \( L \). Then \( BIT_A(n) = \Omega(n \log n) \).

Proof: Let \( A \) be any algorithm that recognizes \( L \). The information state of a processor after an execution of \( A \) contains its initial value and all the messages (in their order and their direction) that it received or sent during this execution. Let \( S = \{IS_1, \ldots, IS_s, \ldots \} \) be the set of all distinct information states. This means that for every \( IS_i \) there exists a word \( w \), such that there is an execution of \( A \) on the ring labeled with \( w \), upon which termination there is a processor with information state \( IS_i \). If the set \( S \) is finite, then \( BIT_A(n) = |S| n = O(n) \) and \( L \) is regular (see Section 5 for bidirectional rings), hence we assume \( S \) is infinite.

We define an infinite set of words \( W = \{w_1, \ldots, w_l, \ldots \} \) as follows: \( w_l \) is a shortest word for which there exists an execution \( ex_1 \) of \( A \), upon which termination on the ring labeled \( w_1 \), there exists a processor \( p \) with information state \( IS_1 \). Assume that after the execution \( ex_1 \) of \( A \) on \( w_1 = \sigma_1 \cdots \sigma_n \), there exist three processors \( p_j, p_k, p_l \) and \( j < k < l \), with the same information state. If the processor \( p' \) lies between \( p_j \) and \( p_l \) (including \( p_j \)) then there exists an execution \( ex_j \) of \( A \) on \( w_j' = \sigma_1 \cdots \sigma_j \cdots \sigma_{k-1} \sigma_k \cdots \sigma_{l-1} \sigma_l \), such that all the processors will have the same information state as in \( ex_1 \) (including the processor \( p' \) that will have the information state \( IS_j \)). The execution \( ex_j \) on \( w_j' \) can be viewed as the execution \( ex_j \) on \( w_j \), where the
segment between \( p_j \) and \( p_k \) is not visible, and only the messages sent from and into if participate in the execution. In other words, in every segment the segment will consist of only one processor (namely \( p_j \)), that will send and receive the same messages as the entire segment. Since both \( p_k \) and \( p_j \) have the same information state, the communication between \( p_j \) and \( p_{j-1} \) is identical to that between \( p_k \) and \( p_{k-1} \); therefore, neither \( p_{j-1} \) nor \( p_k \) will be able to note the difference (between \( w_j \) and \( w_k \)), hence also no other processor will be able to note the difference. This contradicts the assumption that \( w_j \) is a shortest word for which there exists an execution, after which there is a processor, with information state \( IS_j \). If the processor \( p' \) lies between \( p_j \) and \( p_k \) then there exists an execution of \( A \) on \( w'_j = \sigma_1 \sigma_2 \cdots \sigma_k \), which will result in a similar contradiction. Therefore, for every word in the set \( W \), when the execution of \( A \) terminates, at most two processors have the same information state. In order to encode \( \lceil \frac{n}{2} \rceil \) distinct information states we need at least \( \Omega(n \log n) \) bits, hence \( BIT_A(n) = \Omega(n \log n) \).

We also proved that if an algorithm \( A \) has bit complexity \( BIT_A(\lambda) = O(\lambda) \) then \( \lambda \) is finite. From this we derive two important conclusions:

**Corollary 3:** Let \( A \) be an algorithm with \( BIT_A(n) = O(n) \). Then \( A \) uses a finite number of messages (on all possible rings). Namely, there exists \( c > 0 \), such that \( |M_A| < c \).

**Corollary 4:** Let \( A \) be an algorithm with \( BIT_A(n) = O(n) \). Then the maximal number of messages that a processor sends in any execution is bounded. Namely, there exists \( c > 0 \), such that \( n_A < c \).

5. **REGULAR LANGUAGES - BIDIRECTIONAL RINGS**

In this section we extend the results of Section 3 from unidirectional to bidirectional rings:

**Theorem 5:** Let \( L \) be a regular language. There exists a bidirectional algorithm \( A \) that recognizes \( L \) and satisfies \( BIT_A(n) = O(n) \).

**Proof:** Follows immediately from Theorem 1. \( \Box \)

**Theorem 6:** Let \( L \) be a language, that is recognized by a bidirectional algorithm \( A \) with \( BIT_A(\lambda) = O(\lambda) \). Then \( L \) is regular.

**Proof:** Let \( A \) be a bidirectional algorithm that recognizes \( L \), such that \( BIT_A(n) = O(n) \). We construct from \( A \) an equivalent unidirectional algorithm \( A' \) such that \( BIT_A(n) = O(n) \). The construction is done in two stages.

**Stage 1:** From \( A \) we construct an equivalent algorithm \( A' \), that performs the computation on a bidirectional line of processors \( p_1, p_2, \cdots, p_n \) (with the exception of the first message). Every processor \( p_i \) (\( 1 < i < n \)) on the line can communicate with \( p_{i+1} \) and \( p_{i-1} \). \( p_1 \) can communicate only with \( p_2 \) and \( p_n \) can communicate only with \( p_{n-1} \). At the initialization of \( A' \), the leader \( p_1 \) sends a message to \( p_n \), informing it that it is the end of the line (this message is not considered part of \( A' \)). When \( p_1 \) or \( p_n \) want to communicate with each other, they do it through the line \( p_1, p_2, \cdots, p_n \). All the other processors, besides passing these messages, execute the algorithm \( A \).
We assume that $BIT_A(n) = O(n)$. By Corollary 4 there is a constant $c_1$ such that no processor sends more than $c_1$ messages in any execution of $A$. By Corollary 3 the number of distinct messages used by $A$ is bounded by some constant $c_2$. The number of messages added for the communication between $p_1$ and $p_n$ is bounded by $2c_1n$. We have to distinguish between messages sent between $p_i$ and $p_i$, and the other messages, and this can be done by adding one bit to each message, thus the number of bits in a message is bounded by $1 + \lceil \log c_2 \rceil$. The number of bits in the messages sent by $p_2 \cdots p_{n-1}$ is bounded by $2BIT_A(n)$. The bit complexity of $A'$ therefore satisfies $BIT_A(n) \leq (2c_1(1 + \lceil \log c_2 \rceil))h + 2BIT_A(n)$, hence $BIT_A(n) = O(n)$.

For every execution in $A$ there is a corresponding execution of $A'$. If the leader accepts (rejects) in $A'$, then in the corresponding execution in $A$ the leader accepts (rejects); hence $A'$ and $A$ are equivalent.

Stage 2: From $A'$ we construct an equivalent unidirectional algorithm $A''$. The algorithm is divided into passes. At each pass the leader tries to check whether it could have terminated in an accepting information state $IS$ (an accepting information state is an information state (see proof of Theorem 4) such that when the leader terminates it in it, it accepts the pattern on the ring). Initially the leader sends an accepting information state $IS_1$ to $p_2$. Each processor $p_i$, $2 < i \leq n$, receives from $p_{i-1}$ the set of all possible information states of $p_{i-1}$. The processor $p_i$, $1 < i \leq n$, computes, for each such information state, the set of all its consistent information states (an information state $IS_j$ is consistent with an information state $IS_i$, if there is an execution of the algorithm, after which there are two consecutive processors the first in information state $IS_i$ and the second in information state $IS_j$) and sends this set to $p_{i+1}$ (note that the set could be empty). The last processor $p_n$ receives the set of possible information states of $p_{n-1}$ and checks whether there is at least one information state that is consistent with any of them, in which case it sends a "success" message to the leader, otherwise it sends a "failure" message. If the leader receives a "success" message it accepts and terminates, else it checks another accepting information state. If there are no more accepting information states it rejects and terminates.

If the leader accepts in $A''$, it will accept in every execution of $A''$ and terminate in an accepting information state. One of these accepting information states will be detected in $A''$ and cause the leader to accept. If the leader rejects, there is no accepting information state, for which there is an execution of $A''$, and therefore the leader will reject in $A''$. $A'$ and $A''$ are therefore equivalent.

By Corollary 3 there is a constant $c_3$ that bounds the number of distinct messages used by $A'$, and by Corollary 4 there exists a constant $c_4$ that bounds the number of messages a processor sends in any execution of $A'$. The number of bits of each message is bounded by $c_3 \lceil \log c_2 \rceil$, hence each pass uses $O(n)$ bits. The number of passes is bounded by $c_4$. Therefore $BIT_A(n) = O(n)$, and by Theorem 3 it follows that $L$ is regular.

Remark: Stage 1 is essential for the proof, otherwise there will be no processor that will be able to decide if the communication is possible. There are examples in which every two consecutive information states are consistent, but there is no execution of the algorithm, in which their combination occurs.
6. MISCELLANEOUS

We conclude with few notes about the bit complexity hierarchy, and the trade-off between the number of passes and the bit complexity for regular languages.

(1) There is a linear language \( L \) (see [HUI]), such that every algorithm \( A \) that recognizes \( L \) satisfies \( \text{BIT}_A(n) = \Omega(n^2) \).

Let \( L = \{ x \mid x = w c w', w \in \{a,b\}^* \} \). Every letter in \( w \) should be compared with the corresponding letter in \( w' \), which implies the lower bound of \( \Omega(n^2) \) bits.

(2) There is a context-sensitive language that is not context-free (see [HUI]), and that can be recognized in \( O(n \log n) \) bits.

The language \( L = \{ 0^n 1^n 2^n \mid n > 0 \} \) can be recognized in \( \tilde{O}(n \log n) \) bits, using three counters sent around the ring.

(3) For every function \( g(n) \), \( \Omega(n \log n) < g(n) < O(n^2) \), there exists a language such that every algorithm \( A \) that recognizes it satisfies \( \text{BIT}_A(n) = \Omega(g(n)) \), and there is an algorithm \( A \) that recognizes \( L \), satisfying \( \text{BIT}_A(n) = \tilde{O}(g(n)) \).

Let \( \Sigma \) be any alphabet, and \( L_\Sigma = \{ w \mid \exists x, y \in \Sigma^*, i > 0, \text{such that } w = x^i y, |x| > |y| \text{ and } \left\lfloor \frac{g(1)|w|}{n} \right\rfloor \right\rfloor = |x| \} \). If \( i > 0 \), so for every \( n \) there is at least one word \( w \in L_\Sigma \), such that \( |w| = n \).

Any algorithm that recognizes \( L_\Sigma \) requires \( \Omega(|x| + |y|) \) bits. Even if we assume that \( n \) is known to every processor, and that every processor knows which bit of \( x \) it holds, still at least \( n - |x| - |y| \) of the processors will send \( |x| \) bits for \( w \in L_\Sigma \) (otherwise, there will be two distinct subwords of size \( |x| \) for which one of these processors will send the same sequence of bits, and clearly one of these subwords yields a word not in \( L_\Sigma \) that will be accepted, a contradiction). Therefore we get the lower bound of \( \Omega(g(n)) \) bits.

The following algorithm recognizes \( L_\Sigma \) in \( O(g(n)) \) bits: The leader computes \( n \) (using \( O(n \log n) \) bits), and then determines \( |x| = \left\lfloor \frac{g(1)|w|}{n} \right\rfloor \), and compares every segment of length \( |x| \) with the next segment (using \( O(|x| + n) = O(g(n)) \) bits). Therefore \( \text{BIT}_A(n) = O(g(n) + n \log n) = O(g(n)) \). It follows that the language \( L_\Sigma \) requires \( \Theta(g(n)) \) bits for its recognition.

(4) If \( n \) is known then no gap exists.

The language \( L_\Sigma \) defined above will do. Note that since \( n \) is known there is no need for the \( O(n \log n) \) bits to compute it, and for the same reasons as in (3), for every function \( g(n) \), \( \Omega(n) < g(n) < O(n^2) \), the language \( L_\Sigma \) requires \( \Theta(g(n)) \) bits for its recognition. Furthermore, there are in this case non-regular languages that can be recognized in \( O(n) \) bits.

(5) This note concerns the number of bits vs. the number of passes in recognition of regular languages, in a unidirectional ring. We show a language that requires \( c n \) bits in two passes and \( 2^c n \) in one pass.

Let \( \Sigma = \{ \sigma_0, \sigma_1, \ldots, \sigma_2^c-2 \} \), and \( L = \{ w \in \Sigma^* \mid |w|, \text{mod}(2^c-1) \text{ appears an even number of times in } w \} \). If we
want to recognize $L$ in two-passes we can do it with bit complexity of $(2k + 1)n$. In the first round we check $|w| \mod (2^k - 1)$ (using $k$ bits), and in the second pass we send the result (using $k$ bits) and an extra bit to check whether $\sigma_{1, w \mod (2^{k - 1})}$ appears an even number of times. In one round we need to check the parity of all the $\sigma_i$s concurrently, hence we need $(k + 2^k - 1)n$ bits ($k$ bits for $|w| \mod (2^k - 1)$, and one bit for each $\sigma_i$, $0 \leq i \leq 2^k - 2$). It is easy to extend this example and show that the above scenario is the worst possible in the following sense: if a regular language $L$ can be recognized with bit complexity that is bounded by $cn$ (for some constant $c$) and any number of passes (this number; by Corollary 4, has to be bounded), then $L$ can be recognized by a one-pass algorithm with bit complexity bounded by $2^n$.

7. SUMMARY AND OPEN PROBLEMS

In this paper we studied the bit complexity of language recognition on a bidirectional and unidirectional ring with a leader. We showed that a language is recognized in $O(n)$ bits if and only if it is regular, and that a non-regular language requires at least $\Omega(n \log n)$ bits. The range of $\Theta(n \log n)$ to $\Theta(n^2)$ still remains unexplained, but it was shown that it does not correspond to the Chomsky hierarchy. From our results it follows that only regular languages can be recognized without the knowledge of $n$, and for every non-regular language we can assume that $n$ is known.

In the introduction we mentioned similar results for Turing machines. It is shown in ([HA],[HE],[F]), that a one-tape Turing machine recognizes a language in time $O(n)$ if and only if the language is regular, and that a non-regular language requires at least $O(n \log n)$ time. One might argue that every algorithm $A$ can be transformed into a Turing machine $TM_A$ in the following way:

1. The set of messages $M_A$ will be the set of states of $TM_A$.
2. The transition function is the function that from an information state and an input message creates the output message.

Two main problems arise:

1. The set $M_A$ can be infinite.
2. Even if the set $M_A$ is finite the the transition function can be in infinite.

Because of these two problems the transformation of an algorithm $A$ into a Turing machine $TM_A$, such that the bit complexity of $A$ equals the time complexity of $TM_A$, is not straightforward. An example is an algorithm $A$ that counts the number of processors in one pass: clearly $A$ uses $O(n \log n)$ bits. Although there exists a $TM$ that performs the same task in $O(n \log n)$ moves, there is no simple way of constructing the $TM$ from the algorithm, so that the bit complexity equals the time complexity. On the other hand, given a $TM$ with time complexity $t(n)$, one can transform it into an algorithm $A$ such that $BIT_A(n) \leq t(n) \log |Q|$, where $Q$ is the set of states of $TM_A$. Using this transformation, the $\Omega(n \log n)$ time complexity for the recognition of a non-regular language by Turing machines can be shown to be implied.
by our $O(n \log n)$ bound for the bit complexity of the recognition of this language.

We list a few open problems:

(1) Characterize the non-regular languages by their bit complexity (the range $\Theta(n \log n)$ to $\Theta(n^2)$).

(2) How does the knowledge of $n$ affect the results?

(3) Given a regular language $L$, construct an optimal algorithm that recognizes $L$.

(4) How do the results change if we let $\Sigma$ be infinite?

(5) Given an algorithm $A$ with an infinite set of messages, does there exist an equivalent algorithm $A'$ that uses a finite set of messages, and have the same bit complexity?

REFERENCES


