Acta Informatica 36, 447–462 (1999)

Memory requirements for silent stabilization*

Shlomi Dolev[1,**], **Mohamed G. Gouda**[2], **Marco Schneider**[2]

[1] Department of Mathematics and Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel (e-mail: dolev@cs.bgu.ac.il)
[2] Department of Computer Science, The University of Texas at Austin, Austin, TX 78712-1188, USA (e-mail: {gouda,marco}@cs.utexas.edu)

Received: 16 December 1996 / 13 August 1999

Abstract. A stabilizing algorithm is *silent* if starting from an arbitrary state it converges to a global state after which the values stored in the communication registers are fixed. Many silent stabilizing algorithms have appeared in the literature. In this paper we show that there cannot exist constant memory silent stabilizing algorithms for finding the centers of a graph, electing a leader, and constructing a spanning tree. We demonstrate a lower bound of $\Omega(\log n)$ bits per communication register for each of the above tasks.

1 Introduction

The large number of processors and the often unreliable communication media of a distributed system force its implementors to look for a fault-tolerant design. A desirable property is automatic recovery following the occurrence of faults. Automatic recovery is guaranteed when the system is designed to be *stabilizing* [5, 25]. A stabilizing distributed system converges to a desired behavior starting from any state.

In this research we define and investigate a specific class of stabilizing algorithms, namely *silent* stabilizing algorithms. A stabilizing algorithm is *silent* if it converges to a global state after which the values stored in the communication registers are fixed. While some problems are inherently non-silent: e.g. mutual exclusion over a network, other problems can afford silent

* An extended abstract of this paper was presented in the fifteenth annual ACM Symposium on Principles of Distributed Computing, [7].

** Partly supported by the Israeli ministry of science and arts grant #6756195. Part of this work was done while visiting the department of computer science, Texas A&M university. Partly supported by NSF Presidential Young Investigator Award CCR-9396098.

or non-silent solutions: e.g. leader election. Many problems that require stabilization have elegant silent solutions.

Beyond the simplicity implied by the silence property, a silent algorithm may utilize less communication operations and communication bandwidth. Following the convergence stage of a silent stabilizing algorithm, processors need only verify that the value of the communication registers are not changed — thus write operations may be totally eliminated. Moreover, when message passing is used to deliver the value of the communication registers (port buffers) as described in [9] it would be enough to send some encrypted proof that the value is not changed. At each delivery a key is chosen randomly and the checksum relatively to this key is sent together with the key. In this way the communication bandwidth usage can be dramatically reduced.

The interest in distributed and parallel systems of (identical) processors with small memory size is motivated by current microprocessor technology (See e.g., [22], [1]). The mass production of microprocessors motivates multiprocessing systems in which each processor is equipped with a small amount of memory. Even in case the memory size of the microprocessors is not too restricted, the communication bandwidth, or the size of the communication port, and in fact the number of wires connected to a communication port, is a matter of serious concern in devising massive parallelism machines[1]. In another context, microprocessors are used in switches of high-speed networks to support the distributed coordination (See e.g. [22]). Ideally, the amount of memory that such microprocessors are equipped with, is small.

This research examines the memory requirements of silent stabilizing algorithms to achieve several fundamental tasks, including finding the centers of a graph, leader election, and spanning tree construction. There is a class of tasks that are inherently non-silent. An example of such a task is mutual-exclusion or token passing where the contents of the communication registers must be changed over time e.g. [10, 11, 18, 23].

In order to prove our lower bounds we assume the existence of a silent legitimate global state and use this state to construct a silent global state that is illegitimate[2]. Silence is a property of the system following the convergence to a legitimate global state. The means by which the silent legitimate global state is reached are not utilized in our proofs[3]. Thus, our lower bound results apply to deterministic, nondeterministic and randomized stabilizing

[1] We thank the anonymous referee for bringing this motivation to our attention.

[2] A similar approach has been proposed in [18] for proving the memory required to avoid silence (deadlock in the case of mutual exclusion), also see [24] for different techniques for proving impossibility results in uniform systems.

[3] In contrast see [12] for several impossibility results based upon convergence.

algorithms for synchronous systems, asynchronous systems and systems that are controlled by a central-demon (See e.g. [5]).

We consider *uniform*, *semi-uniform* and *id-based* systems as we now define informally (See [10]). In a *uniform* system all the processors with the same number of neighbors are identical, in a *semi-uniform* system all the processors are identical except for a single distinguished leader, and in an *id-based* system each processor has a unique identifier. Processors communicate between each other through shared communication registers. All registers have an identical number of bits. The number of bits in the communication registers from which processors read and write implies a lower bound on the local memory required per a processor — processors need to store the communication register values. To make our lower bound results stronger we prove the lower bounds for the minimal number of bits in the communication registers.

The first task we consider is finding centers of the communication graph. The *eccentricity* of a node in a communication graph is the largest distance from the node to any other node in the graph. A node with minimum eccentricity is called a *center* of the graph. A simple silent stabilizing algorithm for finding the centers of trees is presented in [21]. The algorithm uses $\Theta(\log n)$ memory per processor. Our first result shows that at least $\Omega(\log n)$ bits per communication register (and hence per processor) are required by any silent stabilizing algorithm to find the centers of an arbitrary graph. This result also applies to the restricted case of trees and thus the algorithm in [21] is optimal in its memory requirements.

Recently, relatively intricate randomized algorithms for leader election with a small amount of memory were proposed by [3] and [19]. Our research shows that every silent leader election algorithm for arbitrary graphs requires $\Omega(\log n)$ bits per communication register in a uniform system. This result also applies to uniform id-based systems where processors are augmented with unique identifiers. In the id-based system processors have at least $\log n$ bits for the representation of their identifier. Nevertheless, even in such a case it is still unclear whether $\Omega(\log n)$ bits have to be communicated through the communication registers. Our results show this to be the case.

Constructing a spanning tree of the communication graph in a stabilizing distributed fashion is addressed in e.g. [10], [2], [13, 14] and [26, 15]. The constructed trees are used in [10] for achieving mutual-exclusion, in [2] for performing distributed reset, in [13, 14] for routing virtual circuits with maximum bandwidth and in [26, 15] for maximizing arbitrary routing metrics. For a communication graph of diameter d, the stabilizing spanning tree construction for semi-uniform systems, presented in [10] requires $\Theta(\log d)$ bits of memory (note that n is an upper bound for d). The stabilizing spanning tree construction in [2] is for id-based systems in contrast to the others and

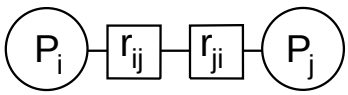

Fig. 1. Edge (i, j) of the communication graph

includes the election of a distinguished leader in order to construct the tree. All three algorithms (as well as the algorithms in e.g. [6,4,8]) benefit from the *silence* property: following the convergence, no change to the value of the communication registers takes place.

A relatively intricate deterministic algorithm for constructing a spanning tree in a semi-uniform system using constant size communication registers was presented in [16]. The algorithm of [16] is not silent. Our results show that at least $\Omega(\log n)$ bits per processor are required for any silent stabilizing spanning tree construction algorithm.

The reminder of the paper is organized as follows. In the next section we present our definitions. In Sect. 3 we discuss our proof method. In Sect. 4, Sect. 5 and Sect. 6 we present the lower bounds for finding the centers, leader election and tree construction, respectively. We present our conclusions in Sect. 7.

2 Silent stabilizing systems

A distributed system consists of n processors denoted by $P_1, P_2, \cdots, P_n$. Each processor in a system resides on a distinct node of the system's *communication graph* $G = (V, E)$. Two processors connected by an edge of G are *neighbors*. Communication among neighboring processors is carried out by *communication registers*. In the sequel we use the term registers for communication registers. An edge $e = (i, j)$ of G stands for two registers $r_{i,j}$ and $r_{j,i}$. For every such edge $e = (i, j)$, P_i and P_j can read the contents of $r_{i,j}$ and $r_{j,i}$. P_i (P_j) can also write into $r_{i,j}$ ($r_{j,i}$, respectively). We refer to the registers in which P_i writes as P_i's registers.

The *environment* of P_i consists of its local topology in the communication graph and the set of registers it can read from. P_i's set of readable registers consists of its own (writable) registers and the (readable) registers that its neighbors share with it (see Fig. 2.).

The *state* of P_i consists of its internal variables.

Configuration: Denote by S_i the set of states of P_i, and denote by R_i the set of values that can be stored in the register r_i. A *configuration*, $c \in (S_1 \times S_2 \times \cdots S_n \times R_1 \times R_2 \cdots R_m)$ of the system is a vector of states of all the processors, and the values of all the registers. In this paper we consider rings, chains, and trees of processors and for each such particular graph we use a simplified representation. For instance, in a chain of processors $P_1, \cdots, P_n$,

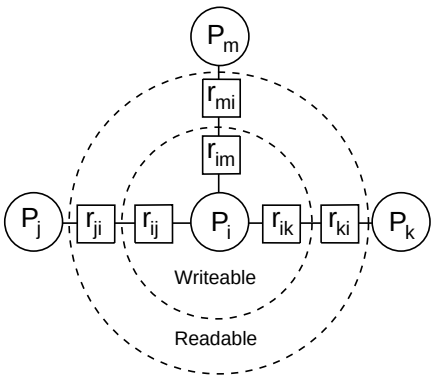

Fig. 2. Environment of P_i

each processor P_i, $1 \leq i \leq n$, can communicate with its left neighbor P_{i-1} (not applicable to P_1) and its right neighbor P_{i+1} (not applicable to P_n). Between any two processors P_i and P_{i+1} lie their communication registers $r_{i,i+1}$ and $r_{i+1,i}$. The configuration of a chain is represented as $(S_1 \times R_{1,2} \times R_{2,1} \times S_2 \times \cdots R_{n-1,n} \times R_{n,n-1} \times S_n)$.

Transition: Each processor processor P_i has a *transition function* (that may be defined by a program executed by P_i). The *transition function* of a processor is a set of *actions*. Each action of a processor P_i can be executed only when the state of the processor and the content of its registers and the neighbors registers have predefined values (that are chosen for this particular action). An *action* of P_i changes the state of P_i and the values of one or more of its own (writable) registers based upon the current state of P_i and the values of one or more of its readable registers.[4] A *transition* is a pair of configurations (c, c') such that executing a subset of all the processor actions at c yields c'. [5]

Run: A *run* is an infinite sequence of configurations $c_1, c_2, \cdots$ such that each pair (c_i, c_{i+1}) is a transition. Each configuration c_i is said to be *reachable* from the initial configuration of the sequence,c_1.

Legitimate and illegitimate configurations: Implicit in the design of any system is a labeling of its configurations as *legitimate* or *illegitimate* such that every configuration that is reachable from a legitimate configuration is

[4] This definition permits P_i to read the registers from one or more neighbors and write to any of its own registers in one action. A more common definition is that an *action* of P_i may include only one communication operation **read** or **write** (See [10]). Our particular choice for defining an *action* implies stronger impossibility results — we consider only a subset of the possible interleaving of the communication operations and still succeed in proving the memory size lower bounds.

[5] Again the choice of this definition strengthens our negative results because it allows multiple levels of concurrency.

itself a legitimate configuration. We identify as *legitimate* those configurations which occur under the correct (intended) execution of a system. All other configurations are considered *illegitimate*. It is up to a system designer to determine which configurations are legitimate and which are not. For example, in a token passing system, clearly any configuration with more than one token would be illegitimate.

Stabilizing and silent stabilizing: A system is called *stabilizing* iff each run of the system has a legitimate state. A configuration c is *silent* iff each configuration that is reachable from c has the same values for its communication registers as in c. A stabilizing system is *silent* iff its silent configurations are the same as exactly its legitimate configurations.

In this paper we consider three types of distributed systems. We define two processors as identical if they both have the same set of states and the same set of actions. In a *uniform system* all the processors that have the same number of neighbors are identical, are identical and the subscripts $1, 2, 3, \cdots n$ are used for convenience only. In a *semi-uniform* system all the processors but P_1 are identical. P_1 may have a different set of states and a different set of actions. Again the subscripts $2, 3, \cdots n$ are used for convenience only. We refer to P_1 as the distinguished leader in a semi-uniform system. Finally, in an *id-based* system each processor, $P_i, 1 \leq i \leq n$ has a unique identifier and the actions of P_i are parameterized by this unique identifier. The unique identifiers in id-based systems are randomly chosen from a large set of identifiers, a set that is much larger than n, hence no algorithm can assume that a particular identifier exists in the system.

The focus of this paper is determining the memory requirements of silent stabilization in relationship to the size of a system. We started off this section by defining a distributed system to consist of n processors, each of which lies on a distinct node in a communication graph. Because we are interested in memory requirements, we will need to consider system constructions that work for arbitrary size n and for arbitrary communication graphs that fit the constraints of the task at hand. We will use the term *algorithm* to refer to such constructions.

3 Proving the memory requirements for silent stabilization

In each of the theorems in this paper we utilize a common proof method to obtain our results. A proof of a theorem: "The number of memory bits per register of any silent stabilizing algorithm for doing X is $\Omega(\log n)$" proceeds as follows. First we assume there is a silent stabilizing algorithm for doing X using $o(\log n)$ bits per processor. Under this assumption we construct one or more legitimate silent configurations of arbitrarily large size for that algorithm. We then show how to combine processors from the

legitimate silent configurations to construct an illegitimate configuration of arbitrarily large size. If the environment of each processor remains the same, the resultant configuration will be silent and illegitimate. Since all silent configurations are legitimate by the definition of a silent stabilizing system, we have a contradiction.

4 Centers of a graph

A simple silent stabilizing algorithm for finding the centers of trees is presented in [21]. The algorithm is designed for uniform systems and uses $\Theta(\log n)$ memory bits per a processor. In this section we show this is a tight lower bound for any such algorithm in a uniform system. We also show that any algorithm for finding the centers of a graph in an id-based or a semi-uniform system requires $\Omega(\log n)$ memory bits per communication register as well.

Our impossibility results are proven for a chain (i.e. line) of identical processors. We prove that the size of a communication register is at least $\Omega(\log n)$ bits. This lower bound implies that $\Omega(\log n)$ bits is a bound on the memory size of a processor that reads the contents of the registers.

Theorem 4.1 *The number of memory bits per register of any silent stabilizing centers finding algorithm for uniform systems is* $\Omega(\log n)$.

Proof. Assume that there exists a silent stabilizing centers finding algorithm for uniform systems. Denote the number of values that can be stored in a register by k.

Consider a chain of n processors where a processor P_i, $1 \leq i \leq n$, can communicate with its left neighbor P_{i-1} (not applicable to P_1) and its right neighbor P_{i+1} (not applicable to P_n). Let $c = x_1, a_1, b_1, x_2, a_2, b_2, \cdots, x_i, a_i, b_i, \cdots, x_n$ be a silent configuration of the chain where x_i is the state of the processor P_i and a_i (b_i) is the value stored in $r_{i,i+1}$ ($r_{i+1,i}$, respectively).

Assume $n = k^2 + 2$. We claim that there exist i, and j, $i \neq j$, such that $a_i = a_j$ and $b_i = b_j$. Consider that the number of possible combinations for a pair (a_i, b_i) is k^2. Thus out of $k^2 + 1$ register pairs, at least one pair of registers (a_i, b_i) appears more than once. In a chain of n processors there are $n - 1$ register pairs. Solving for $n - 1 = k^2 + 1$ we get $n = k^2 + 2$.

We use the notation $c = L_1, a_i, b_i, L_2, a_i, b_i, L_3$ to denote a silent configuration of a chain in which their exists i and j such that $a_i = a_j$ and $b_i = b_j$. In the above notation L_i represents a sequence of processor states and register values that is a portion of the configuration. Recall that a *silent* system configuration is a configuration such that any run that starts in this configuration is silent. Construct the configuration $c' = L_1, a_i, b_i, L_2, a_i, b_i, L_2, a_i, b_i, L_3$

of a system with less than $2 * n$ processors. The environment of every processor in c' is kept unchanged with respect to c, i.e. each processor P in c' reads the same values in its neighboring registers in c' as it does in c. Therefore, no processor writes a new value to its registers and c' is a silent configurations.

Without loss of generality assume that the number of processors in c is even. Otherwise we can assume $n = k^2 + 3$. Since the number of processors in c is assumed to be even, c' either contains more than two centers (when L_2 contains a center) or the centers are not located at the center of the system represented by c' (when L_2 does not contain a center). Let n' be the number of processors in c'. In either case we have a contradiction based upon the assumption that $n' \geq 2(k^2 + 3)$. Thus, in a uniform system of n processors, each register must be able to store at least $k > (n/2 - 3)^{1/2}$ values. Hence, the number of bits in each register is $\Omega(\log n)$. ■

Since a chain is a type of tree as well as a type of arbitrary graph the above theorem applies to trees as well.

Theorem 4.2 *The number of memory bits per register of any silent stabilizing centers finding algorithm for id-based systems is* $\Omega(\log n)$.

Proof. To prove the impossibility result for id-based systems we need to ensure that no identifier appears twice in the silent configuration we construct. In particular, c' cannot be a valid configuration of an id-based system since the processors in L_2 of Theorem 4.1 appear twice. The result is attained as follows. We use two configurations c_1 and c_2 of chain systems of $k^2 + 2$ processors each, and we require that the set of processor identifiers in c_1 does not include any processor identifier in c_2 (and vice versa). We choose c_1 and c_2 out of infinitely many possible systems of $k^2 + 2$ processor chains such that $c_1 = L_1, a_i, b_i, L_2, a_i, b_i, L_3$ and $c_2 = L'_1, a_i, b_i, L'_2, a_i, b_i, L'_3$. To apply the arguments of Theorem 4.1 we define $c' = L_1, a_i, b_i, L_2, a_i, b_i, L'_2, a_i, b_i, L_3$ ■

Theorem 4.3 *The number of memory bits per register of any silent stabilizing centers finding algorithm for semi-uniform systems is* $\Omega(\log n)$.

Proof. For the case of semi-uniform system we cannot assume that c' and c'' are valid configurations since it is possible that one of the processors in L_2 is the special processor. In such a case c' includes two special processors and c'' includes none. To apply the arguments of Theorem 4.1 we choose c out of those system configurations in which the special processor is the leftmost processor in the chain. ■

5 Leader election

Recently, relatively intricate stabilizing randomized algorithms for leader election with a small amount of memory were proposed by [3] and [19]. These algorithms are not silent. Our research shows that every silent stabilizing leader election algorithm for arbitrary graphs requires $\Omega(\log n)$ bits per communication register in a uniform or id-based system. Leader election in a semi-uniform system is a trivial task that does not require communication — the special processor is a natural candidate for a leader.

In an id-based system, it is clear that processors must be equipped with $\Omega(\log n)$ bits to store their unique identifiers. Still it is not clear whether the size of the communication registers of a silent stabilizing leader election algorithm can be constant. Our lower bound implies that even for id-based systems the size of the registers must be logarithmic in the number of processors. Thus, for instance, there is no algorithm with constant size registers that (somehow, say serially) communicates the identifiers and then converges to a silent configuration.

Stabilizing leader election algorithms for prime sized uniform rings with constant memory per processor are presented in [20]. The algorithm in [20] assumes that the activities in the system are controlled by a central demon [5]. The central demon activates a single processor at a time. Note that the algorithm of [20] cannot be applied to a synchronous system since symmetry cannot be broken if all processors simultaneously execute an action [5]. We also show that regardless of the level of concurrency, there is not a silent stabilizing leader election algorithm for primed sized rings with constant memory per processor.

We prove our lower bound for a ring of processors. We prove that the size of a communication register is at least $\Omega(\log n)$. This implies a bound of $\Omega(\log n)$ on the memory size of a processor.

Theorem 5.1 *The number of memory bits per register of any silent stabilizing leader election algorithm for uniform or id-based systems is* $\Omega(\log n)$.

Proof. Assume that there exists a silent stabilizing leader election algorithm. Denote the number of values that can be stored in a register by k.

Consider a ring of n processors, where a processor P_i, $1 \leq i \leq n$, can communicate with its left neighbor P_{i-1} (the left neighbor of P_1 is P_n) and its right neighbor P_{i+1} (the right neighbor of P_n is P_1). Let $x_1, a_1, b_1, x_2, a_2, b_2, \cdots, x_i, a_i, b_i, \cdots, x_n, a_n, b_n$ be a silent configuration of the system, where x_i is the state of the processor P_i, a_i (b_i) is the value stored in $r_{i,i+1}$ ($r_{i+1,i}$, respectively).

Assume $n = k^2 + 1$. We claim that there exist i and j, $i \neq j$, such that $a_i = a_j$ and $b_i = b_j$. Consider that the number of possible combinations for a pair (a_i, b_i) is k^2. Thus out of $k^2 + 1$ register pairs, at least one pair of pair

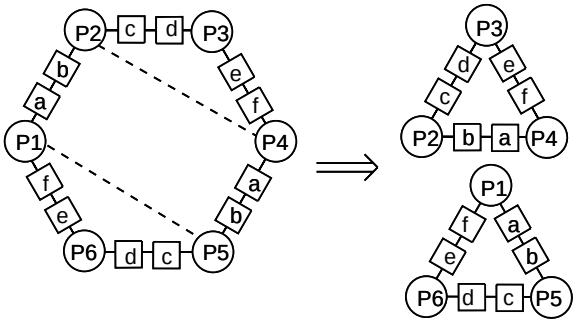

Fig. 3. Constructing two rings from one

of registers (a_i, b_i) appears more than once. In a ring of n processors there are n register pairs and thus $n = k^2 + 1$.

We use the notation $c = L_1, a_i, b_i, L_2, a_i, b_i$ to denote a silent system configuration of a ring in which there exist i and j such that $a_i = a_j$ and $b_i = b_j$. Construct two configurations $c' = L_1, a_i, b_i$ and $c'' = L_2, a_i, b_i$. If c' and c'' correspond to rings then it is easy to see that they are silent configurations.

It is possible however that either c' or c'' does not correspond to ring. This can only happen if the two register pairs are not separated by three or more processors. In order to ensure that the register pairs are separated by three or more processors we can consider a ring of size $n \geq 6(k^6 + 1)/2$. We claim that in every ring of size $n \geq 6(k^6+1)/2$ there exist two identical sequences of six register values that do not overlap. The claim is proved by counting the number of possible combinations for a six-tuple which is k^6. To ensure one six-tuple appears twice we need to accommodate $6(k^6 + 1)$ registers in ring. This requires $6(k^6+1)/2$ processors. Thus, in case $k \leq (2n/6-1)^{1/6}$ there must exist two identical and non-overlapping sequences of six register values. This will ensure that the register pairs are separated by three or more processors (see Fig. 3. for an example).

We may now proceed by case analysis. If the leader is in L_1 then c'' is a silent configuration with no leader. Otherwise, c' is a silent configuration with no leader.

Since the construction preserved the uniqueness of id's, we have a contradiction for the case of a uniform system and the case of an id-based system. Our contradiction is based upon the assumption that $n \geq 6(k^6+1)/2$. Thus in a uniform or id-based system of n processors, each register must be able to store at least $k > (2n/6-1)^{1/6}$ values. Hence the number of bits in each register is $\Omega(\log n)$. ■

The result in [20] is for rings with a prime number of processors. In Theorem 5.1 we have used one legitimate silent configuration to construct

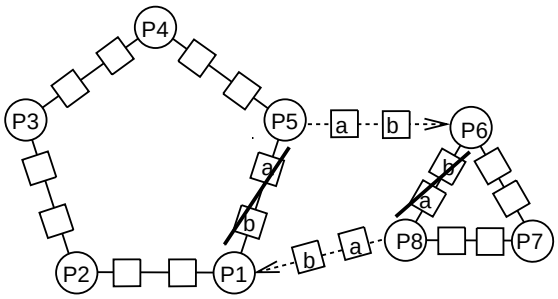

Fig. 4. Combining two rings

another illegitimate silent configuration for a ring of a different (smaller) number of processors. If the constructed configuration represents a ring with a composite number of processors then the result of Theorem 5.1 is not applicable to the algorithm presented in [20]. In the next Theorem we prove that there is no constant size stabilizing silent algorithm for the restricted case of prime sized uniform rings.

Theorem 5.2 *There is no silent stabilizing leader election algorithm for prime sized uniform rings with a constant number of bits per register.*

Proof. Let p_1 and p_2 be the number of processors in two rings $R1$ and $R2$, respectively. Dirichlet's Theorem (See e.g. [17]) states that any arithmetic progression $a + bn$ with gcd(a,b) = 1 contains infinitely many primes. Thus for infinitely many z, $p_1 + zp_2$ is prime.

Using the silent configurations of rings of size p_1 and p_2 we show how to construct a silent configuration for a ring of size $p_1 + zp_2$ that contains more than one leader.

Consider two silent configurations, c and c', of rings with p_1 and p_2 processors, respectively. If $c = L_1, b_i, a_i, L_2$ and $c' = L_3, b_j, a_j, L_4$ such that $a_i = a_j$ and $b_i = b_j$ then c and c' can be combined as follows: $c'' = L_1, b_i, a_i, L_4, L_3, b_i, a_i, L_2$. The number of processors in c'' is $p_1 + p_2$. Note that there are infinitely many rings of prime size but only k^2 combinations of values for a_i, b_i. Thus, there must be two rings of size $p_1 \neq p_2$ that have silent configurations such as c and c' above.

Figure 4. illustrates the combining of two rings that share a pair of register values. The ring P_1, P_2, P_3, P_4, P_5 is combined with the ring P_6, P_7, P_8 to produce the ring $P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8$.

The silent configuration of a ring of size p_2 may be combined with itself arbitrarily many times to produce a silent configuration of zp_2 processors using the above method. The resulting silent configuration may be combined with the silent configuration of a ring of p_1 processors to produce a silent configuration of $p_1 + zp_2$ processors. Such a silent configuration will include

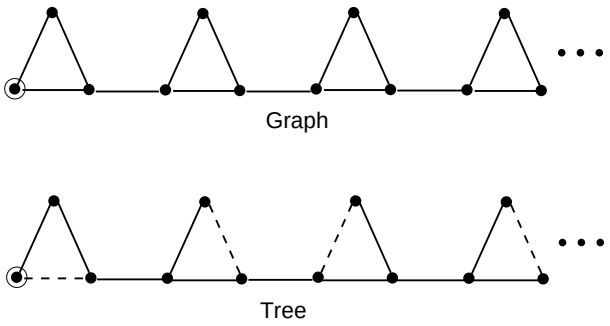

Fig. 5. Triangle graph and its tree

$z + 1$ leaders. We have proved our contradiction, since for infinitely many values of z the above silent configuration represents a ring with a prime number of processors. ■

6 Tree construction

Silent stabilizing algorithms for tree construction can be found in [10], [2], [13, 14] and [26, 15]. Each of these algorithms uses $\Theta(\log n)$ bits per register. The algorithms in [10], [13, 14] and [26, 15] are designed for semi-uniform systems. The algorithm in [2] is designed for an id-based system and includes the election of a distinguished leader in order to construct the tree. In this section we prove that there is no silent stabilizing algorithm for tree construction in semi-uniform or id-based systems that uses $o(\log n)$ bits per register. Hence we prove that that each of these algorithms is optimal in its memory requirements.

Theorem 6.1 *The number of memory bits per register of any silent stabilizing tree construction algorithm for uniform, semi-uniform and id-based systems is* $\Omega(\log n)$.

Proof. In the figures that follow filled dots represent processors and lines represent communication links (implemented by shared communication registers). The distinguished leader, P_1, is marked by a circled dot.

We consider a special graph which is formed by a sequence of triangles wherein each pair of consecutive triangles is connected by an edge (See Graph of Fig. 5.). In a spanning tree there is exactly one path from the root to every processor. Thus, exactly one link of every triangle in the graph does not belong to the tree (See Tree of Fig. 5.).

Denote the number of values that can be stored in a register by k. In every graph of $n > k^2 + 2$ triangles, a silent configuration will contain two non-triangle edges with the same pair of register values. This is shown in the top part of Fig. 6. labeled “Tree with Distinguished Leader”.

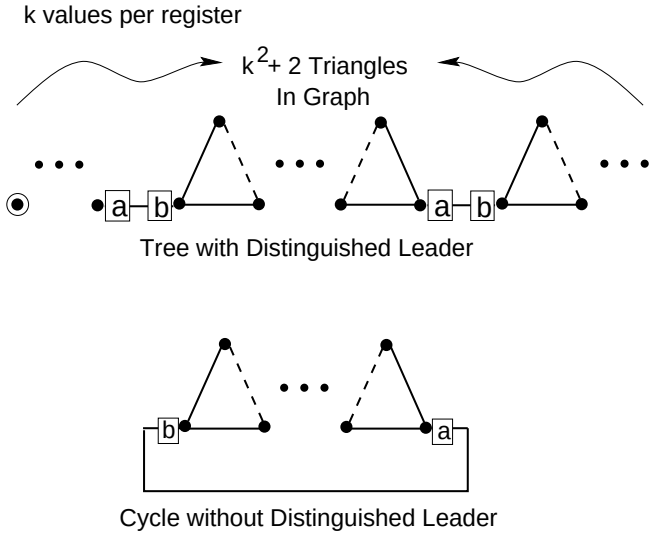

Fig. 6. Tree with leader and cycle without leader

We use the tree to construct a cyclical graph without a distinguished leader. Let r_{1a} and r_{1b} be the pair of registers with values a and b on the left side of Fig. 6., and let r_{2a} and r_{2b} be the registers marked by a and b on the right side of the Fig. 6.. The construction "disconnects" r_{1a} from r_{1b} and also "disconnects" r_{2a} from r_{2b}. It then "connects" r_{1a} with r_{2b} and r_{2a} with r_{1b}. This construction preserves the environment of every processor in the resultant graph. Consider that the processor that read the value a from r_{1a} in the tree graph reads the value a from $r_{j,j+1}$ in the cycle graph. Similarly, the processor that read the value b_i from $r_{j+1,j}$ in the tree graph reads the value b_i from $r_{i+1,i}$ in the cycle graph. At this stage we can conclude the lower bound for uniform and id-based systems[6] — the obtained configuration of the cycle graph is a silent configuration and each processor has a unique id in the case of an id-based system. However, for semi-uniform systems we must show that a distinguished leader exists in the silent configuration. To do so we perform an additional construction step.

The last step is illustrated in the *Non-Tree with Distinguished Leader* graph of Fig. 7.. In this step the tree with a distinguished leader and the cycle without a distinguished leader are combined to obtain a non-tree (cycle containing) graph with a distinguished leader. The idea is to disconnect a non-tree link from a triangle in both graphs and to connect the tree and the

[6] The astute reader will have foreseen this result for the case of rooted tree construction. Rooted tree construction in a system without a distinguished leader provides an implicit leader election.

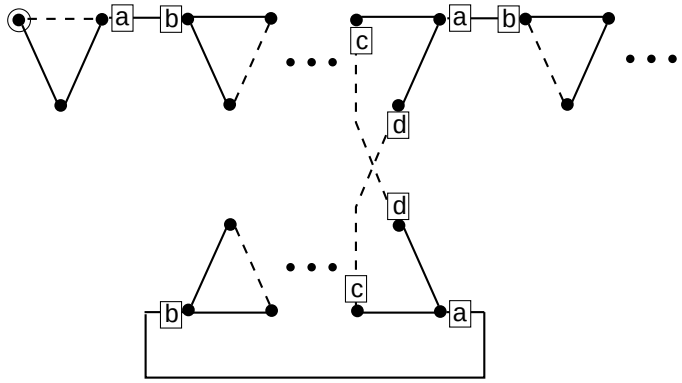

Fig. 7. Final semi-uniform construction

non-tree (by non-tree links) in a way that ensures that the environment of every processor is kept unchanged.

Given a graph of $k^2 + 2$ or more triangles, and its silent configuration corresponding to a tree, we have constructed a silent configuration corresponding to a non-tree. The construction can be applied to any graph with more than k^2+2 triangles. Since the number of processors in the constructed graph is $\Theta(n)$, the order of k is $\Omega(n^{1/2})$. Thus, the number of memory bits required is $\Omega(\log n)$. ■

Finally we note that it is possible to consider an id-based system with a distinguished processor (i.e. semi-uniform id-based). A slight modification of the proof for the above theorem may be applied to such a system. The combined tree with the leader and the non-tree without a leader must have processors with different identifiers, similar to the technique used in Theorem 4.2.

7 Concluding remarks

We defined the class of silent stabilizing algorithms and proved that any centers finding, leader election and tree construction algorithms in this class require $\Omega(\log n)$ bits per communication register and processor.

In order to prove our lower bounds we assumed the existence of a legitimate silent configuration and used this configuration to construct an illegitimate silent configuration. The means by which the legitimate silent configuration was reached were not utilized in our proofs. Thus, our results apply to deterministic, nondeterministic and randomized stabilizing algorithms for synchronous systems, asynchronous systems and systems that are controlled by a central-demon. In addition, our proofs did not rely on

the atomicity level of actions. Thus our results apply to any level of action atomicity.

Our results are also applicable to the case where processors can read the entire state of their neighbors as proposed in [5]. In such a setting once the system reaches a silent (safe) configuration no processor changes its state. If there exists a silent algorithm for, say, finding the centers of a chain that uses $o(\log n)$ bits per processor state, then an algorithm in which processors write their entire state in the registers must also be silent — this is impossible by our lower bound.

There are silent stabilizing algorithms that require a constant number of bits per register, one such example is the graph coloring algorithm of [8]. The classification of tasks according to their memory requirements for silence configuration may identify the level of *locality* of the task. Roughly speaking, a silent algorithm that uses $o(\log n)$ bits per register does not support a global structure such as: the existence of a *single* center (or two centers), the existence of a *single* leader or *single* tree. We believe that further research for lower bounds on the memory requirements of other silent stabilizing algorithms and the design of such algorithms will lead to important elegant and practical stabilizing algorithms.

References

1. James Abello, Shlomi Dolev: On the Computational Power of Self-Stabilizing Systems. J. Comput. Inf. **1**(1), 585–603 (B10), 1994. Theor Comput Sci **182**, 159–170 (1997)
2. Anish Arora, Mohamed Gouda, Distributed Reset. IEEE Transact. Comput. **43**(9) (1994)
3. Baruch Awerbuch, Rafail Ostrovsky: Memory-Efficient and Self-Stabilizing Network RESET. Proc. of the 13th Annual ACM Symp. on Principles of Distributed Computing, pp. 254–263, 1994
4. Zeev Collin, Shlomi Dolev: Self-Stabilizing Depth First Search. Inf. Proc. Lett. **49**, 297–301 (1994)
5. Edsger W. Dijkstra, Self-Stabilizing Systems in Spite of Distributed Control. Commun. ACM **17** (11), 643–644 (1974)
6. Shlomi Dolev: Optimal Time Self Stabilization in Dynamic Systems. Proc. of the 7th International Workshop on Distributed Algorithms, pp. 160–173, 1993. J. Parallel Distrib. Comput. **42**, 122–127 (1997)
7. Shlomi Dolev, Mohamed G. Gouda, Marco Schneider: Memory Requirement for Silent Stabilization. Proc. of the 15th Annual ACM Symp. on Principles of Distributed Computing, pp. 27–34, 1996
8. Shlomi Dolev, Ted Herman: SuperStabilizing Protocols for Dynamic Distributed Systems. Proc. of the Second Workshop on Self-Stabilizing Systems,UNLV, pp. 3.1–3.15, 1995. Short abstract in Proc. of the 14th Annual ACM Symp. on Principles of Distributed Computing, p. 255, 1995. Chicago J. Theoret. Comput. Sci. **3**(4) (1997)
9. Shlomi Dolev, Amos Israeli, Shlomo Moran: Resource Bounds for Self Stabilizing Message Driven Protocols. Proc. of the 10th Annual ACM Symp. on Principles of Distributed Computing, pp. 281–293, 1991. SIAM J. Comput **26**(1), 273–290 (1997)

10. Shlomi Dolev, Amos Israeli, Shlomo Moran: Self Stabilization of Dynamic Systems Assuming Only Read Write Atomicity. Distributed Comput. **7**, 3–16 (1993)
11. Mohamed G. Gouda, Furman Haddix: Stabilizing Token Rings in Three Bits. J. Parallel Distrib. Comput. (to appear 1996)
12. M. G. Gouda, R. R. Howell, L. E. Rosier, The instability of self-stabilization. Acta Informatica **27**, 697–724 (1990)
13. Mohamed G. Gouda, Marco Schneider: Stabilization of Maximum Flow Trees. Invited talk: Proceedings of the Third Annual Joint Conference on Information Sciences, November 1994. A full version is in preparation for submission to IEEE Transactions on Parallel and Distributed Systems
14. Mohamed G. Gouda, Marco Schneider: Maximum Flow Routing. Proc. of The Second Workshop on Self-Stabilizing Systems, UNLV, pp. 2.1–2.13, 1995
15. Mohamed G. Gouda, Marco Schneider: Stabilization of Maximal Metric Trees. Proc. of The Fourth Workshop on Self-Stabilizing Systems, IEEE, June 5, 1999
16. Colette Johnen, Joffroy Beauquier: Space-Efficient Distributed Self-Stabilizing Depth-First Token Circulation. Proc. of the Second Workshop on Self-Stabilizing Systems, UNLV, pp. 4.1–4.15, 1995
17. Hua Loo Keng: Introduction to Number Theory, p. 243. Berlin, Heidlberg, New York: Springer, 1982
18. Amos Israeli, Marc Jalfon: Token Management Schemes and Random Walks Yield Self Stabilizing Mutual Exclusion. Proc. of the 9th Annual ACM Symp. on Principles of Distributed Computing, pp. 119–131, 1990
19. Gene Itkis, Leonid Levin: Fast and lean self-stabilizing asynchronous protocol. Proc. of the 36th Annual IEEE Symp. on Foundations of Computer Science, 1994
20. Gene Itkis, Chengdian Lin, Janos Simon: Deterministic, Constant Space, Self-Stabilizing Leader Election on Uniform Rings. Proc. of the 9th Workshop on Distributed Algorithms, 1995
21. Mehmet Hakan Karaata, Sriram V. Pemmaraju, Steven C. Bruell, Sukumar Ghosh: Self-Stabilizing Algorithms for Finding Centers and Medians of Trees. Proc. of the 13th Annual ACM Symp. on Principles of Distributed Computing, pp. 374, 1994
22. A. Mayer, Y. Ofek, R. Ostrovsky, M. Yung: Self-Stabilizing Symmetry Breaking in Constant-Space. Proc. 24th ACM Symp. on Theory of Computing pp. 378–382, 1992
23. G. Parlati, M. Yung: Non-Exploratory Self-Stabilization for Constant-Space Symmetry-Breaking. Proc. 2nd Annual European Symposium on Algorithms, ESA, pp. 183–201, 1994
24. Sandeep K. Shukla, Daniel J. Rosenkrantz, S. S. Ravi: Observations on Self-Stabilizing Graph Algorithms for Anonymous Networks. Proc. of the Second Workshop on Self-Stabilizing Systems, UNLV, pp. 7.1–7.15, 1995
25. Marco Schneider: Self-Stabilization. ACM Computing Surveys, Vol. 25, No. 1, March 1993
26. Marco Schneider: Flow Routing in Computer Networks. Ph.D. Dissertation, Department of Computer Sciences, The University of Texas at Austin, December 1997. URL: http://www.cs.utexas.edu/users/marco